ALLEN N. SMITH
Editor and Program Chairman

DON B. MEDLEY
Conference Chairman

AFIPS PRESS
1815 NORTH LYNN STREET
ARLINGTON, VIRGINIA 22209

AFIPS CONFERENCE PROCEEDINGS

1983 NATIONAL COMPUTER CONFERENCE

May 16–19, 1983
Anaheim, California

Library of Congress Catalog Card Number 80–649583
ISSN 0095-6880
ISBN 0-88283-039-2

AFIPS PRESS
1815 North Lynn Street
Arlington, Virginia 22209

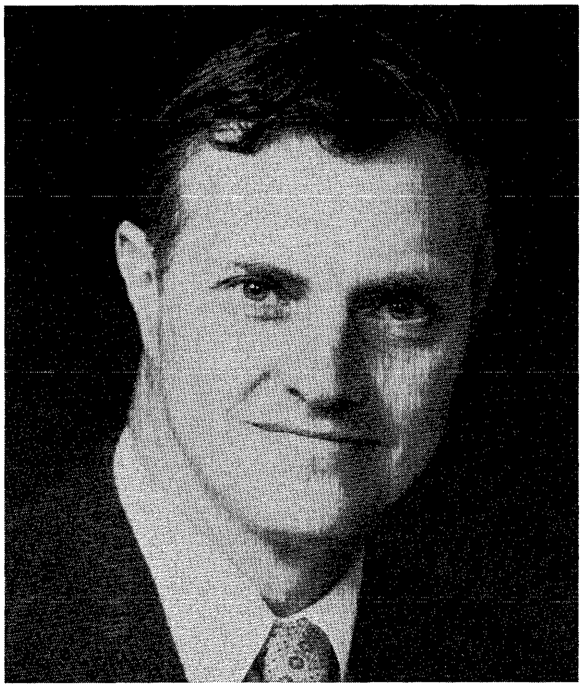
Printed in the United States of America

DON B. MEDLEY
1983 NCC Conference Chairman

Preface

A very bright and dedicated group of computing professionals has labored long and hard to develop a high-quality technical program for the 1983 NCC. These *Proceedings* represent a printed record of many of the presentations planned for the conference educational program. This collection should provide valuable reference material to computing professionals in years to come.

In addition to the papers included in the *Proceedings* and presented at the 1983 NCC, many panelists and speakers also participate, making a complete NCC educational program. I hope that you will find time to attend one or more of these valuable sessions and that this volume of the *Proceedings* will be a useful source of information for you in later years.


ALLEN N. SMITH
1983 NCC Program Chairman

Introduction

"The Emerging Information Society: Computers, Communications, and People" is the theme of the 1983 National Computer Conference. It became very clear as the *Proceedings* developed that this theme is most appropriate for the conference. Many of the sessions within program tracks interweave papers, presentations, and ideas that relate to other tracks. Many papers in the *Proceedings* could have been placed in several different tracks. The computing field is now clearly a multidisciplinary field. Thus the *Proceedings* place emphasis on office systems, personal computers, telecommunications, and human factors. Of even greater significance is the growing need to integrate all these disciplines to understand the entire field today. It is our hope that the 1983 *Proceedings* provide such a viewpoint.

With this theme as a foundation, the NCC '83 *Proceedings* have been grouped in nine major areas:

1. Software Engineering—providing new ideas and directions for the development of systems in the future.
2. Management/Education—a broad view of various management and education issues, with a mini-track on maintenance of systems, a growing problem in the field.
3. Database/Distributed Systems—an update on new trends in software and hardware for database management.
4. Human and Social Issues—a broad coverage of issues relating to the impact of computing on society, organizations, and the individual.
5. Office Automation—an update on many facets of office automation, including the growing impact of personal computers on this area.
6. Decision Support Systems—a significant new area of growth in providing executive and professional support is covered in these sessions.
7. Hardware—a view of the trends and developments in new approaches to computer hardware and architecture.
8. Telecommunications/Applications—a series of updates on new and recent developments in telecommunications along with brief updates on a series of applications and on the use of the technology.
9. Personal Computers—a broad view of the explosively growing area of microcomputers, both for personal and for business use.

Pioneer Day will focus on Howard Aiken and the Harvard Computational Laboratory, tracing early developments in the computer field.

As with the program for 1982, we have reduced the number of sessions to 84, whereas past programs gave more than 100. This has allowed us to concentrate the program and the *Proceedings* on the areas of greatest importance and to offer higher quality. We have selected over 80 papers from the vast number of papers that were submitted; we believe the ones selected provide high quality in their areas. We had to eliminate many fine topics and decline many fine papers. The conference program includes a key to the page numbers of the papers in the *Proceedings* for easy reference. Summaries of the panel discussions are not printed in the *Proceedings;* however, a brief summary of each track area is included in the *Proceedings.*

The development of the 1983 NCC program required the dedicated effort of many individuals: the Program Committee members, the session organizers and leaders, the panelists and presenters, and authors of technical papers, as well as the referees who helped us select the papers to be presented in the *Proceedings.* Additionally, the staff of AFIPS greatly assisted us in developing the *Proceedings.* The Committee staff, headed by Jeff Young, with the invaluable assistance of Carrie Borgen and Georgia Marinelli, contributed more than it is possible to acknowledge to the development of this volume. I wish to extend my own personal thanks to all of these individuals, and especially to the Program Committee. It is our sincere hope that this program will provide useful and thought-provoking knowledge to those who attend.

CONTENTS

SOFTWARE ENGINEERING

Toni Shetler
TRW
Redondo Beach, California

The Software Engineering track at this year's NCC has twelve sessions that are rich with ideas and information; included are: methodology, technology, tools, management, user considerations, and research and development.

Software engineering research and development activity is addressed by three separate panels from the research, defense, and industrial communities. Experts will examine the critical priorities, technologies, issues, and resources instrumental in directing current R&D efforts and in formulating the ideas and identifying the concerns that will be part of our future.

Software development techniques are addressed in four sessions that focus on available methods and tools. Methods for improving development productivity through techniques such as reusable code will be presented. Presented will be improvements in existing programming methodologies including development support environments, testing and debugging tools, as well as effective documentation methods.

Software development management is addressed by a panel experienced in managing large projects and delivering state-of-the-art software systems. They will discuss their experiences and work environments, and project these experiences into the future, touching on the issues that software development management will face in the years to come.

Artificial intelligence is addressed in two sessions. One session explores experiences with systems that adapt to user interface requirements. The other session provides a panel of AI experts to explore the value of AI in the tools of the future. These sessions provide a refreshing look at how AI has evolved from the fantasy fringe to the mainstream of computing.

The two Ada™ sessions address the present and the future: early experiences developing Ada applications and Ada programming environments. The software engineering community has watched Ada evolve during the 70's—from concept to definition to early implementation. This evolution had more care, planning, coordination, and tracking than any of its language predecessors. These sessions provide an opportunity to learn about the Ada activity, thinking, and directions for the 80's and beyond.

"Communicating with Databases in English": This session introduces currently available systems for asking questions of databases in English and describes commercial user experiences with them. The focus is on methods of enabling users to redefine the subset of English appropriate for their own applications and to switch easily from one context to another. The session was organized by the Association for Computational Linguistics to present the best current work in applied natural language processing.

"Artificial Intelligence: Blue Sky or Tools of the Future?" This session presents work from several points across the research and development spectrum, from work characterized as "why-do-you-want-to-do-THAT?" to work whose solid, practical results are being used in everyday industrial and academic applications.

"Writing Less Code—An Approachable Ideal": This session is composed of three reviewed papers that are included in the *Proceedings*.

"Software Management for the 80's": A group of experienced, large-scale software development managers present their experience and intuition, addressing the problems and solutions of managing software development projects in the changing computer hardware and software technological environments. This session will concentrate on large-scale systems. The panelists discuss past experiences in software management given changes in technologies, what the panelist is doing to manage effectively at present, and predictions for the future.

"Reducing Program Development Risks with Reusable Code": The need for improved programmer productivity has spurred the development of a variety of program generation aids. One significant approach to applications development is reusable COBOL code modules. The developers and users of Raytheon's ReadyCode discuss the theory of reusability and its implementation in specific user environments.

"Software Engineering Techniques and Approaches": This session is composed of five reviewed papers that appear in the *Proceedings*.

"Directions in Software Engineering: Now and The Future": This panel session discusses the recent workshop held by ACM SIGSOFT and IEEE Technical Committee on Software Engineering concerning activities they could sponsor to speed the development of software engineering techniques and their successful transfer into practice. Recommended actions from the workshop are discussed and followed by open discussion.

"Software Technology for Adaptable Reliable Systems": A discussion of the new DOD software initiative program is presented. Two panelists, heavily involved in its formulation, discuss the issues.

"Experience in Ada™ Applications": This session focuses on some of the first projects using Ada™ as an implementation language. Each of the speakers has been involved with a large Ada application. In particular, this session provides their early lessons learned, as well as describes how Ada was introduced to their organizations.

"Future Visions: Ada™ Environments of the 1990's": This session is composed of two reviewed papers and a panel of three. The two papers appear in the *Proceedings;* the panelists, involved in Ada™ environment research review the issues.

"Software Engineering by the Year 2000": This panel of distinguished software engineers explore the directions software engineering might take in the next 17 years. Panelists present their predictions, followed by an interchange among panelists and with the audience.

"Effective Software Documentation—Online Documentation": Online documentation is replacing hard-copy, or written, documentation for computer systems and software. This session explores the past, present, and future of online documentation.

Writing less code—An approachable ideal

by NAOMI LEE BLOOM
American Management Systems, Inc.
Arlington, Virginia

ABSTRACT

We are being inundated by a sea of unsatisfied user expectations. This growing, and sometimes frightening, backlog of application development requests has been much discussed but little reduced. One almost universal approach to reducing this backlog has been to try to improve the productivity of our scarce technical resources (programmers, systems analysts, etc.). A more promising approach to meeting user application needs may be to substantially reduce the amount of new code needed to satisfy these needs. It takes no great insight to become convinced that, other things being equal, the less code written to achieve a specific level of systems support, the less risk, cost, elapsed time, and frustration must be accepted by the organization. This paper presents a brief survey of some common, and some less obvious, applications-enabling techniques. Two of the most promising techniques, foundation software and adaptable application packages, are more fully described in separate papers.

INTRODUCTION

We are being inundated by a sea of unsatisfied user expectations. This growing, and sometimes frightening, backlog of application development requests has been much discussed but little reduced. And the invisible backlog, described by Martin[1] as the unspoken (and perhaps not yet dreamed of) desires of our users, ensures that this problem is not likely to diminish.

One almost universal approach to reducing this applications backlog has been to try to improve the productivity of our scarce technical resources (programmers, systems analysts, etc.). Productivity techniques, such as structured programming, structured analysis, regression testing, and interactive programming, have been widely adopted, but still the backlog grows. Clearly, even quantum leaps in the productivity of scarce technical resources will not eliminate this backlog.

A more promising approach to meeting user application needs may be to substantially reduce the amount of new code needed to satisfy these needs. Such application-enabling techniques, to use a phrase that seems to have originated within IBM, are intended to reduce the amount of new code written rather than to merely expedite the production of new code. It takes no great insight to become convinced that, other things being equal, the less code written to achieve a specific level of systems support, the less risk, cost, elapsed time, and frustration must be accepted by the organization.

It is important to note, however, that other things are usually not equal. Many of the techniques described in this paper substitute increased consumption of computing resources for reductions in the personnel resources needed to achieve a certain level of user support. As hardware costs and the resulting price performance ratios continue to improve while competent analysts, programmers, and related computer professionals grow more scarce and more expensive, it is a reasonable business judgment to explicitly trade off increased hardware resource consumption for man-hours of development and user time. Such trade-offs must not compromise satisfying user needs and they must be carefully evaluated for each application so that the system overheads associated with various packages and tools do not catch the project team unawares.

This paper presents a brief survey of some common and some less obvious applications-enabling techniques. Two of the most promising techniques, foundation software and adaptable application packages, are more fully described in separate papers by Curtis[2] and Woodward and DiGiammarino.[3] If properly used, the techniques presented here will reduce not only the amount of new code written by any one organization, but also the aggregate amount of new code. However, even if these applications-enabling techniques are fully applied, some new code will have to be written, and that should be done in a highly productive and orderly way. While this paper and those by Curtis and Woodward and DiGiammarino focus mainly on traditional business applications, applications-enabling techniques may be applied equally to the development of scientific, system-oriented or personal applications.

THE SPECTRUM (OR HIERARCHY) OF APPLICATIONS-ENABLING TECHNIQUES

There is nothing very mysterious about finding ways to write less code. You can do any of these things:

1. Convince the user not to want (or to need) a new application.
2. Reuse old code—your own or someone else's.
3. Use simple tools (remember how levers work?) to multiply the work value of any code you do write.
4. Get someone else, perhaps your users, to write the code for you.

The key to successful applications enabling is to build these very simple maxims into your systems development life-cycle methodology. Applications development or even package installation projects should not be initiated, unless the new application is really needed. And in every stage of the life cycle, you must ask yourself what alternatives exist to developing new code. Thus, applications-enabling techniques parallel, in some sense, the applications development life cycle.

In the earliest stage, frequently called the business systems or strategic systems planning stage, you must ask the fundamental question of whether this application is worth doing at all. As the process goes forward, you should be asking the following types of questions:

1. Has this application been developed before? If so, there may be some old code that you can reuse.
2. Does this application lend itself to the use of simple tools? Either tools that someone else has developed, or that you yourself could develop?
3. Does this application lend itself to the end user-written code that is characteristic of many data manipulation and analysis applications?

By asking these types of questions at the appropriate points in the systems development life cycle, you can take advantage of the many techniques available to minimize the amount of new code written. The remainder of this paper explores these techniques in the order in which they tend to present themselves in the life cycle.

DO NOT DEVELOP UNNECESSARY APPLICATIONS!

The most obvious solution to our problem of how to write less code is to eliminate from the backlog all but the essential (translation: justified) applications. Strategic systems planning (also known as business systems planning) is the process by which an organization identifies and prioritizes its major systems development objectives. By explicitly aligning the applications development priorities with the organization's business strategy, we take a critical first step toward reducing the amount of new code to be written.

Although there are many flavors of strategic systems planning described in the literature, the objectives identified by IBM[4] in their business systems planning methodology are representative:

1. To provide management with a formal, objective method for establishing priorities for corporate information systems without regard to local interests
2. To ensure that scarce development resources are committed to those systems that have a long life, thereby protecting the systems investment, because these systems are based on the business processes that are generally unaffected by organizational changes
3. To provide that the data processing resources are managed for the most efficient and effective support of the business goals
4. To increase executive confidence that high-return, major information systems will be produced
5. To improve relationships between the information-systems department and users by providing for systems that are responsive to user requirements and priorities
6. To identify data as a corporate resource that should be planned, managed, and controlled in order to be used effectively by everyone

By ensuring that we develop only those applications whose relevance to the organization and benefits have been rigorously examined, we have made the first breakthrough toward minimizing the backlog of unsupported application requirements. To repeat, if you develop no unnecessary applications, you will not be called upon to write (and maintain!) worthless code.

REUSE OLD CODE—YOUR OWN OR SOMEONE ELSE'S!

Where an application is justified, there are several possibilities for developing it without writing any code or by writing only a small amount of (it is hoped) simple code. Application software packages have been available for nearly 30 years, and many routine business (and system, e.g., sorting) functions are very adequately supported by such packages. In addition, many of your business functions, such as edit routines for specific data elements, have probably been programmed many times within your own organization. Before deciding that an application is so unique as to obviate using any existing code—a common attitude among many in-house analysts and users—consider the many flavors of software packages and reusable in-house code.

Currently available commercial applications software can be divided into three general categories:

1. Traditional software packages, which perform a well-defined set of functions with minimal installation options
2. Contemporary software packages, which perform a well-defined set of functions subject to many table-driven, user-defined, installation-specific options
3. Adaptable software packages, which perform a flexible set of functions subject to many table-driven, user-defined, installation-specific options

Traditional Software Packages

Initially, application packages were really custom software that the developer chose to share, albeit for compensation, with others. Early package vendors often sold their essentially custom systems with minimal documentation and installation support. Installing such a package required the buyer to modify code even to support the most obvious installation-specific requirements, for example, to change report headings to contain the buyer's company name.

The buyer of a traditional software package (and there are many currently being sold) gets some clear benefits: On *short* notice, he is able to obtain and install *debugged* code that performs some *well-defined* set of functions after *minimal* source code modification; and he pays a *far* lower purchase price than he would for equivalent custom development. Needless to say, the italicized adjectives are subject to the buyers' personal evaluation. But, in theory, the risks, cost, elapsed time (and, hopefully, frustration) of purchasing a traditional package are less than in doing the application from scratch.

That's the theory, but the benefits are often not realized in practice. With a traditional package, every user-specific requirement, from report headings and formats to variations on common algorithms, resulted in modifications to foreign (at best) or (more often) incomprehensible and undocumented source code. Although traditional packages remain an appropriate technique for writing less code, their inflexibility can be frustrating.

Contemporary Software Packages

Eventually, modern (that is, scientific) approaches to software design, combined with the recognition that even the most flexible software buyer had some unique requirements, led to a new type of package. Written to be generalized, commercial software products, contemporary packages (my term) have the following:

1. Well-documented source code constructed to provide low-risk user exits, that is, specific points at which user-

written subroutines can be inserted without disrupting the program flow or voiding the vendor's warrantee
2. Reference tables that remove from the source code such frequently customized functions as report headings and, in some cases, formats; message code literals and severity levels; data element names, field lengths, data types, and edit rules, including pointers to other reference tables of valid values and code translations; parameter values, for example, process scheduling dates, current withholding tax percentages, and airline overbooking percentages; calculation algorithms—sophisticated packages exist for which not only the parameter values but also the operators and calculation bases are table-driven; and coding structures, for example, the chart of accounts or organizational structure
3. A formal installation process, including sample conversion programs, job streams, and other code-reducing aids

Like traditional packages, the purchase and use of contemporary application packages generally reduces the costs, risks, elapsed time, and personal frustrations of meeting system support needs. However, there is always a price for flexibility. Sophisticated reference tables can require considerable loading and maintenance effort, although this approach is far less risky than modifying source code. Plus, users can often be roped into taking responsibility for loading and maintaining most of the tables.

More important, from the perspective of containing cost, risk, and elapsed time, the availability of options means someone (usually a cast of thousands) must analyze, document, recommend, evaluate, and (it is hoped) decide on each desired option. But contemporary applications packages go a long way toward meeting organization-specific requirements without developing new code.

One further note before moving into a new area of packaged software. As mentioned earlier, there is usually a hardware resource consumption penalty for using generalized software. Contemporary packages which favor table-driven processes over hard-coded processing, exact a stiffer penalty in this regard than do the traditional packages.

Adaptable Software Packages

One of the most interesting recent developments in software packages is the trend toward building groups of related modules that can be reconfigured to suit various application requirements. One such package was developed to support credit card collection activities (CACS). Recognizing that credit card collections are a specific example of a generic class of applications, that is, case tracking, scheduling, and state-processing functions, the software was developed to automate these generic functions. With a combination of powerful reference tables, including process control tables, and program modules that can be combined in various ways, CACS can be used with minimal source code modifications to support a broad class of user requirements. The paper by Woodward and DiGiammarino[3] describes CACS and the concept of adaptable software in more detail.

Solve Part of the Problem With Old Code

Access to mathematical and statistical subroutines was an early enhancement to many compilers. In contemporary systems, active data dictionaries often drive data element edits from a common or shared subroutine. Indeed, most data processing shops have developed some standard source language components, perhaps as COPYLIB equivalents, that can be reproduced in various applications at minimal risk, cost, and so on. When we discuss using simple tools to leverage the value of any newly written code, one point that we'll develop further is the idea that the design effort must explicitly focus on identifying common processes that could be programmed once rather than needing to be redone in multiple programs or systems or installations.

To take full advantage of existing code (or to identify common processes for initial development), the life-cycle methodology must emphasize answering the following questions at each level of the design:

1. Have we ever automated this function before? Even a relatively minor function, such as a date edit, can be programmed once, even as a generalized routine, at far less cost than having every programmer do his own thing. At a minimum, your effort for the year 2000 will be greatly simplified if you've been smart enough to incorporate a single date routine into all your systems. It's essential to evaluate each process in this way as a potential candidate for the organization's library of standard software.
2. Will we ever need to automate this function again? Date edits, translations of organization codes into their correct names, report headings, and many other common functions appear in nearly every business application. Do them once in a generalized way, at somewhat greater cost initially, and use them forever.

Unless the deliverables at each stage of the development lifecycle explicitly address the issue of standard software (reusable code), many opportunities for writing less code will be missed—now and in the future.

USE SIMPLE TOOLS

There are two general approaches to multiplying the value of any code you do write:

1. Extension software, which uses your (it is hoped) simple code written in the tool's own command language as the input from which it creates (by translation, compilation, assembly or one of several other extension techniques) very substantial functionality; and
2. Conservation techniques, which are a formal set of design techniques that look for the common functional elements in an application in order to develop a single implementation of these common functions for use across the application.

Reusing date routines is a very simple case of conservation. In this section we'll explore more sophisticated examples of the two approaches just mentioned.

Extension Software

When you write JCL to unleash the power of IBM's various operating systems, you are using extension software to minimize the code you must write. My earliest programs in machine language on an IBM 1401 had no such extenders, and we wrote our own tape reads and printer writes. Now, every use of a system utility from within your application, that is, calling the COBOL internal SORT, leverages a few utility commands to perform considerable work.

Thus, the universe of extension software ranges from the old and familiar to the new and still developing:

1. Utility programs that provide system or housekeeping functions
2. Report writers and inquiry languages, including graphics packages
3. Database management systems with which you use simple commands in the application programs to invoke powerful data handling, edit, storage, and access capabilities
4. Screen generators
5. Data management and analysis tools, for example, SPSS and SAS
6. Application generators
7. Very high-level languages.

The boundaries among these tools are not clear-cut, and many of them can be used by nontechnical persons to achieve the ultimate shifting of application development responsibility. All of these tools hold the same promise of providing complex software to leverage simple commands into powerful functionality, and many deliver on this promise.

However, there is a serious fly in the ointment regarding the use of extension techniques. We are now being inundated in a sea of command languages, specialized syntaxes, and easy-to-learn, English-like, languages. There's not even agreement on how commands are delimited! Until considerable standardization occurs, taking advantage of even a small set of these tools will impose a serious training burden on any organization. And many professional programmers and users will resist using these tools because they quite reasonably perceive that the cost of mastering them is too high.

Conservation Techniques

Perceptive analysts and designers have always recognized common functions in their application specifications, but the process of doing so was largely informal. On many business applications, there are a rather large set of common functions that lend themselves to a common software approach. At American Management Systems, Inc., we have incorporated into our life-cycle methodology a quite formal process for searching for these common system elements.

The decision to build an application around a base of common software modules must be made explicit quite early in the design process so that all further effort can be efficiently directed. We call the resulting software, which provides common services to the rest of the application, foundation software. The foundation software approach to developing large application systems is described in detail in the companion paper by Gary Curtis.[2]

GET SOMEONE ELSE TO WRITE THE CODE!

End-user computing is not a new idea. In the beginning of computer history, programming was the adjunct function of scientists, engineers, and mathematicians who were trying to use the great behemoths to calculate ballistic missile trajectories and to develop software for other, equally forbidding problems. In my early days as a programmer, accountants were still developing the first automated payrolls, general ledgers, and banking and insurance systems. Professional programming is less than 20 years old, so why do we now treat end-user computing as a state-of-the-art development?

One reason is that, until now, whoever approached the computer was forced to learn computer-speak—at great personal sacrifice. If we believe the advertisements for various end-user computing tools, the professional programmer may soon focus solely on core production systems and tool development, leaving to the user development of most data extraction and analysis (MIS) systems. But the future has not yet arrived.

Many of the simple tools described in this paper can be used by a nontechnical person after some training, and the growth of information centers attests to the availability of user-friendly tools. Fourth generation languages, for example, RAMIS II or FOCUS, are advertised as powerful tools for developing whole applications from simple commands. The proliferation of personal computers attests to the user orientation of such tools as VisiCalc. Clearly, if the user can directly translate his unspoken (or never clearly spoken) information requirements into a working system, he won't have the DP staff to kick around any more.

CONCLUSION

Computers are worthless without programs, be they software, firmware, or part of the hardware itself. People still write programs, and people are expensive, unpredictable, and fragile. If only to sell more computers, the hardware vendors would welcome (support and probably give birth to) any approach to program development that used more computer resources to free scarce personnel to develop new applications that used more computing resources. Since they develop many of the packages and tools and generally corner the market on really superb professional programmers, software vendors certainly favor the techniques described in this paper. Corporate users and DP managements are also on board the write-less-code bandwagon. So why does the applications backlog continue to grow?

1. In-house programmers would rather write programs (not to mention design whole systems) than load tables for a contemporary package or do report writer setups. Perhaps we need a new category of DP aide or paraprofessional who sees using tools as a desirable job description?
2. Without standardization in grammar or syntax, currently available tools produce a Tower-of-Babel effect wherever they go.
3. Many users have terminal block, not to mention various other phobia, that limit their ability to use any computing tools.
4. Computing resources, while obviously getting less expensive, are *not* free. Their acquisition, which always occurs in large increments, is a more visible expenditure to the organization than is the cost (opportunity cost) of unfulfilled application needs.

Time is clearly on the side of the approaches described in this paper, but I wouldn't yet discharge my COBOL programmers nor declare that all user needs can be satisfied by their new Apples! As in all things, a balanced mix of these new techniques with more traditional application-development strategies will produce the best results.

REFERENCES

1. Martin, James. *Applications Development Without Programmers.* N.J.: Prentice-Hall, Inc., 1982.
2. Curtis, Gary A. "Foundation Software: A Significant Improved Approach To The Development of Large Application Systems." *AFIPS Proceedings of the National Computer Conference* (Vol. 52), 1983.
3. Woodward, Mary, and Peter DiGiammarino. "A Case For Adaptable Applications Software." *AFIPS Proceedings of the National Computer Conference* (Vol. 52), 1983.
4. IBM. "Business Systems Planning—Information Systems Planning Guide," GE20-0527-3, 1981.

Foundation software: A significantly improved approach to the development of large application systems

by GARY A. CURTIS
American Management Systems
Chicago, Illinois

ABSTRACT

The American Management Systems (AMS) approach to the technical framework of large applications systems is based on a concept we call *foundation software,* an integrated environment of standard packages and custom modules that provides common services to the development and operation of applications software. This environment provides a standardized, structured, and simplified view of the outside world to applications software. Use of foundation software dramatically improves the economics of development, operation, and maintenance of large systems and reduces the risk of developing such systems. In this paper, the foundation software approach is defined and illustrated, with particular emphasis on the relationship of foundation software to the overall architecture of large-scale systems and the impact of foundation software on the application-system development life cycle.

INTRODUCTION

The AMS approach to the technical framework of large applications systems is based on *foundation software.* Use of foundation software dramatically improves the economics of development, operation, and maintenance of large systems and reduces the risk of developing such systems. This paper describes the foundation software concept and our experience using this approach from three perspectives:

- The relationship of foundation software to the overall architecture of large-scale systems is discussed, including foundation software functions and components and the major benefits to developing large-scale systems using the foundation software approach.
- The use of foundation software is an inherent part of the *AMS GUIDE: Methodology,* systems-development life cycle. Foundation software activities in each phase of the systems-development life cycle are described.
- The integration of foundation software and application software is discussed from the viewpoint of the overall system architecture.

THE DEFINITION OF *FOUNDATION SOFTWARE*

Foundation software is an integrated environment of standard packages and custom modules that provides common services to the development and operation of applications software. This environment provides a standardized, structured, and simplified view of the outside world to applications software.

Foundation software increases productivity throughout the development, operation, and maintenance of applications by isolating applications software from changes in the technical components of the computer system and by making those components easier to use.

Despite claims to the contrary, most operating systems, database management systems (DBMSs), teleprocessing monitors, and other technical components fail to simplify applications development and operations. Indeed, a common result is that such components serve to greatly complicate the development and operation of applications software. The foundation software approach avoids this complication and delivers the benefits offered by these technical components to application software in a simple, effective manner.

Software Architectural Levels

Large-scale systems can be divided into three major architectural levels. Figure 1 shows the relationships among these

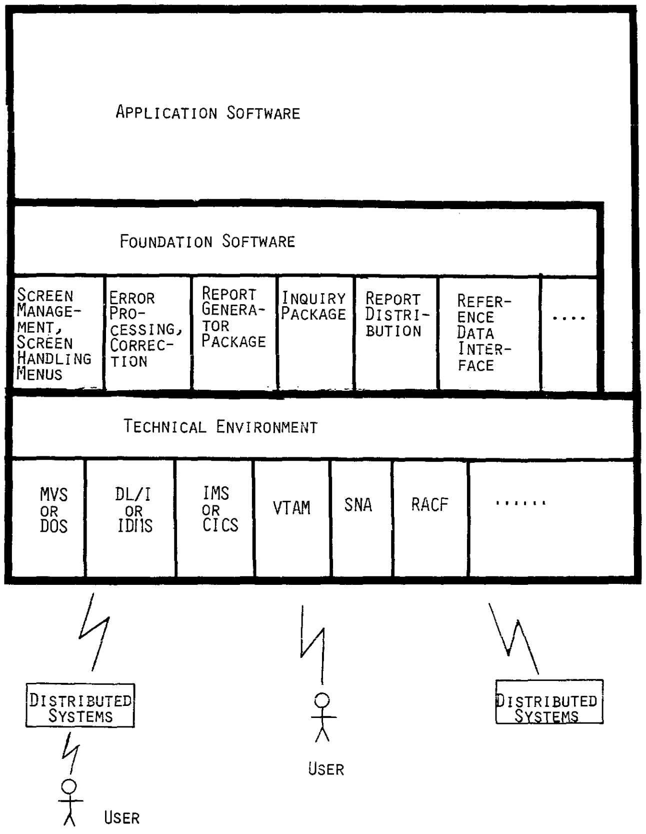


Figure 1—System architectural levels

levels. Associated with each level are specific functional and organizational responsibilities. The levels are

1. *Technical environment*—This level includes the technical components that the system designer takes for granted and that usually cannot be modified for application purposes. It typically consists of
   - system control program (OS/VS, DOS/VSE, MPE, VMS, etc.)
   - access methods and utilities (VSAM, VTAM, IDCAMS, etc.)
   - network architecture (SNA, DECNET, X.25, etc.)
   - telecommunications monitor (IMS DC, CICS, IDMS DC, etc.)
   - DBMS (IDMS, DL/I, Model 204, IMAGE, etc.)

   Responsibility for the maintenance and support of components of the technical environment generally rests with the computing facility and its systems-software organization.

2. *Application software*—This level is the functional core of the application system. It contains all of the specific substantive functions that relate to the application (business) problem at hand:
   - editing, verification, cleansing of application data
   - computations, analysis, transformations of application data.

   Most, if not all, of the application-data-dependent processing operations occur at this level.
3. *Foundation software*—This level provides an interface between the application software and the detailed considerations required by each component of the technical environment. Foundation software directly uses standard programming, communication, and control services of the technical environment, such as database calls, network messages, and control blocks, to provide high-level common application services such as menu processing, security, and error handling. Foundation software may also include software packages such as inquiry software and report generators.

Foundation-Software Functional Scope

The functional scope of foundation software cannot be *rigidly* defined. For any specific application system, determining the functions to be provided by foundation software should be done in the context of the application design characteristics, constraints of the technical environment, and the organizational environment of the system-development effort.

Our experience is that functions with the following characteristics are nearly always more effectively handled as foundation software:

- common use of the function throughout some or all application subsystems
- simplification of complex technical-environment features
- interface to technical-environment features that are subject to a high rate of technological change
- expected volatility in application requirements for the function
- missing or poor technical-environment features
- performance sensitivity.

Although benefits from any one of these attributes can justify inclusion of a function in foundation software, it is usually the case that foundation software functions exhibit benefits due to several of them. Based on these attributes, a general statement of the functional scope of foundation software can be made. The functional areas that foundation software typically comprises are discussed later.

Foundation-Software Components

Foundation software consists of three types of components:

- *Custom interface modules*—Efficient, standardized utilization of the technical environment's most complex components is usually provided through custom interface modules. In some cases, such interfaces provide an entire environment for applications processing. Control relationships among these interfaces and application software vary from normal subprogram linkage to architectures in which application software processes under the control of a foundation software interface environment. Such facilities as a reference-data interface, on-line menu processor, and report distribution subsystem are usually provided in this manner.
- *Packaged software*—Software packages are usually integrated into foundation software with custom interfaces. This topic is considered in more depth later.
- *Common modules*—Processing functions required frequently throughout the application software are provided as common modules that are invoked through standard subprogram linkage. Some functions provided as common modules, such as numeric editing, free-form parsing, and data validation, are generic to most business applications systems. Others provide services that are specific to a particular system or subsystem.

The Role of Packaged Software

Software packages, such as report generators, inquiry packages, and data-entry packages, are usually important components of large applications systems. Packages can provide cost-effective solutions to many of the processing functions of large applications systems. A major difficulty in the effective use of packages in such systems is that packages tend to be functionally narrow in scope and are often cumbersome to use outside the context of standardized interfaces and procedures.

Through the use of foundation software front-end and back-end interfaces, software packages are integrated into the processing environment of large applications systems, such that the functions which the packages provide can be used much more effectively by applications designers and programmers than would be possible if they were used standalone.

For example, in a large-scale IMS DB/DC financial system, ad hoc inquiries into historical and other databases have been provided cost effectively by integrating the INQUIRY IV/IMS query package into the foundation software on-line user environment. Users access the package through the standard application on-line protocol and menus, request their queries in simplified, familiar terminology, and are essentially unaware that a package is being used. This level of integration allows application security controls to be applied to ad hoc queries and allows uncomplicated transfers among package and nonpackage transactions. In this case, functionality was also added to the package through foundation software. The foundation software interface that captures query requests for transfer to the package first scans the requests for search conditions that will result in unacceptably lengthy on-line database processing and redirects such requests to overnight handling.

For a package to be incorporated into foundation software it should be possible to use the package without any internal modifications. This restriction does not include the use of vendor-supported exits through which the package passes control to other software. Indeed, the use of such exits is a common method for tailoring a package to specific application requirements within the foundation software.

Foundation-Software Benefits

The foundation software approach provides benefits during all phases of the system development process. Herein the major ones are described, by phase.

During system design

Foundation software results in a high level of modularization and is, in this regard, an extension of structured design methodology. Common functions are designed only once and many more functions can be provided by standard, reusable software.

The high degree of isolation from a need for detailed understanding of the technical environment enables application designers to concentrate more effectively on solving business problems.

During system development and implementation

Senior technical staff resources are scarce throughout the computing industry. Foundation software allows the efforts of such staff members to be concentrated in high-payoff areas. This allows technically sophisticated applications to be developed by relatively less sophisticated staff.

During the development of large-scale systems, changes in the technical environment are usually introduced independent of the application development effort. Foundation software isolates application programmers from these changes and results in fewer disruptions, less recoding and greater productivity.

During system support

Rapid change in technology presents the system support staff with a continuous major effort to keep application systems functioning against a moving background. With the foundation-software approach, application programs are isolated from the effects of this change. Application software can be maintained by less sophisticated technical staff.

Due to the extensive use of foundation-software common services, new application functions can be added with minimal impact on the existing system. The application can thus be adapted to changing user requirements more easily and more cost effectively.

INTEGRATION WITH THE SYSTEM DEVELOPMENT PROCESS

The foundation-software approach is an inherent part of the *AMS GUIDE: Methodology,* systems-development life cycle. This section examines some of the key aspects of the foundation-software life cycle and the foundation-software development and support team.

Foundation-Software Life Cycle

Foundation-software is designed, developed, and implemented in a life cycle that is integrated with that of the overall application system. The relationship of the foundation-software and the application-development life cycles is shown in Figure 2. Some of the key aspects of the foundation-software life cycle follow.

- *Overall system architecture*—The first step in the foundation-software life cycle is the development of an integrating framework for the design of the system as a whole. The overall structure for the system, the system architecture, is designed as early as possible. The system architecture defines the user interface with the on-line components of the system, the environment in which the system will be developed, and the operational environment in which the application software will run, and provides a model for effective management use of the system. This step is critical to the successful evaluation of the total system through its design, development, and support phases.

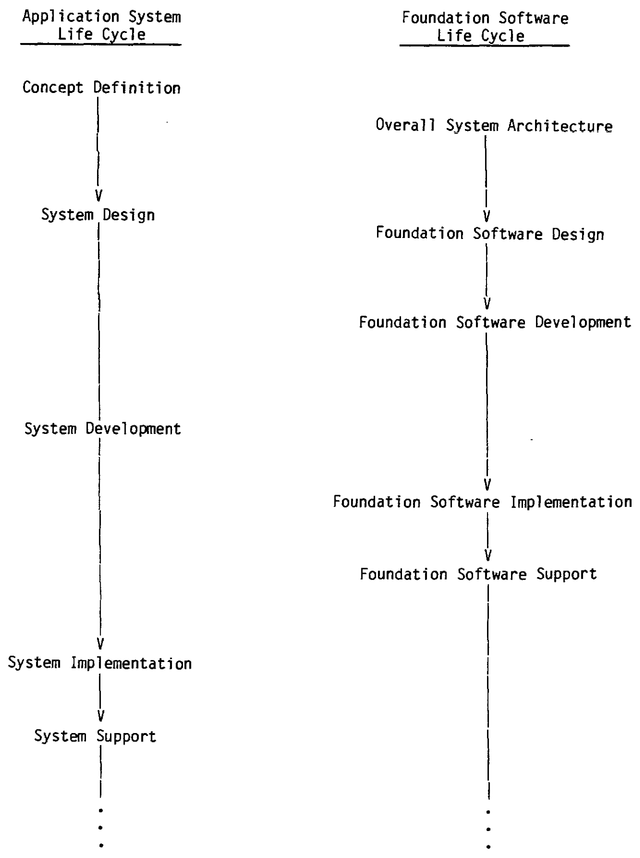

Application System Life Cycle
Concept Definition
System Design
System Development
System Implementation
System Support
Foundation Software Life Cycle
Overall System Architecture
Foundation Software Design
Foundation Software Development
Foundation Software Implementation
Foundation Software Support

Figure 2—Relationship of the foundation software and the application-system life cycles

- *Advanced availability of foundation-software functions*—Foundation-software design begins as early as possible, generally late in the concept definition or early in the system-design phase. This design encompasses all foundation-software components including previously developed foundation software, new custom foundation-software components, and software packages. Foundation-software design specifications must be completed sufficiently early in the system-design phase to support the development of program specifications for the application system. Development of foundation-software begins during application-system design in order to provide working modules, interfaces, and development aids at the beginning of the application system development phase. Foundation-software implementation and support activities begin during application system development in order to ensure stable system-development and turnover activities.
- *Iterative design and development*—Although a basic set of foundation-software functions is designed and developed early in the application-system life cycle, further development of foundation-software functions is an iterative process. The standardized interfaces and functional isolation provided by the foundation-software approach permit experience gained during application development to be fed back into the foundation software without disrupting application development.

Organizational Impact

The system-development project usually has a foundation-software team that is responsible for the design, implementation, and support of all foundation-software components. The primary objective of the foundation-software team's activities is to ensure that foundation-software designs and software are in place sufficiently far in advance of application teams' needs to support the timely progress of the project.

The team consists mostly of foundation-software and technical specialists. The makeup of the team changes as the application system moves through its development phases; that is, in contrast to most application teams, the foundation-software team has some members who bring very specific skills to bear and are needed only until a particular problem is solved. In all other ways, the team is managed as a normal system-development team, reports to the project manager, and is integrated into the project team. The foundation-software team remains a part of the overall project team through implementation and turnover of the entire system, including all foundation-software components.

FOUNDATION-SOFTWARE FUNCTIONS AND FEATURES

The following foundation-software functions reflect AMS experience in various technical environments and application areas and are neither a set of required foundation-software functions nor an exhaustive list. They do, however, represent areas where experience shows the foundation-software approach to be effective and the payoff to be significant.

On-Line User Management

On-line user management facilities provide a friendly, screen-oriented environment that allows the user to exercise all authorized application functions. The language, sequence of operations, and features of this environment are relevant to the application and the user, and are not constrained by jargon and idiosyncracies of the on-line technical environment. Features usually provided include

- menu processing
- screen handling
- security
- user assistance.

Input Management

Input management controls the processing of application data from the point at which the data enter the system until the data have been accepted as valid and has been stored in application-data structures. Data can be entered, processed, corrected, and reprocessed in batch or on-line modes, in any combination. Standardized, application-oriented data organizations such as transaction, document, and batch organizations are used. Specific features include

- data entry
- error processing and suspense
- error correction
- document approval
- input workflow control.

Application Data Management

Foundation software is used to simplify and standardize access to application data structures and, where needed, to make the access more efficient. This function generally takes the form of data-structure (database) access interfaces that present to the application tabular logical views of application data, and of update-isolation facilities which ensure that application data structures are updated in a consistent, synchronized manner. The interfaces manipulate data structures using efficient, and often complex, call patterns, database facilities, and custom-developed functions. The major functions addressed by application data management are

- reference data maintenance and control
- reference data interface
- update isolation
- application data backout.

Network Management

In distributed data-processing environments, application software may operate on different processors connected through a network. In such an environment, isolation from the technical complexity of the network is provided by network-control foundation software. Network-control software allows the application software to be designed and implemented without regard to where in the network architecture the application software must operate. Network control includes the functions of

- network status
- transaction routing
- distributed site support.

Output Management

Management of the varied forms and high volumes of output produced by large-scale application systems is controlled by output-management foundation software. Some of the major functions include

- report distribution
- graphics interface
- report generators
- on-line inquiry.

System Management

Overall management of the processing of a complex application system is simplified through several foundation-software functions. The objective is to present to system administrative personnel a standardized, simplified view of control facilities that makes it possible to exercise complex functions of the technical environment with minimal technical expertise. System-management foundation software provides:

- scheduling
- recovery/restart
- performance monitoring.

Office Automation Facilities

Foundation software integrates application-system data and reports with office-automation facilities in two ways. Where the user's office environment includes existing facilities (such as stand-alone word-processing systems), foundation software provides interfaces that allow application software to send data to and receive it from these facilities. Where office-automation features are required by an application system but are not available in the user environment, foundation software includes both the application interfaces and the actual document preparation and mailbox facilities themselves. The office-automation facilities supported by foundation software include

- document preparation
- word processing
- electronic mail.

Technical-Environment Enhancements

Occasionally, technical environments do not provide some basic system support facilities that are essential to fulfillment of the application's primary objectives. In this situation, the system designer must often decide between a considerable sacrifice in application functionality and the development of significantly more complex application software owing to the incorporation of technical support features. When analysis of this tradeoff leads to a decision to support the application's required functions by developing the complex facilities missing from the technical environment, the foundation software approach minimizes the adverse impact of this additional complexity. It further ensures that the complexity of the application software is not affected. The following are examples of facilities that are normally, and preferably, provided by the technical environment, but that may be provided by foundation software when necessary.

- database locking
- transaction logging
- job control
- dataset management.

FOUNDATION SOFTWARE CASE STUDY

A description of a large-scale integrated financial system implemented under IBM's IMS DB/DC technical environment is presented below. The relationship of the foundation software and application software components are particularly noteworthy. Figure 3 shows the overall system architecture keyed for the following discussion.

The on-line user interface (1) handles user sign-on, sign-off, and security checking. It presents users with a series of menus to get to the desired system function, be it data entry, processing, or an inquiry request. As a security precaution, if a terminal has not been used for an extended period (set by the system administrator—perhaps 15 minutes), then the On-Line User Interface will automatically sign-off the terminal.

A data entry/error correction program (2) accepts input transactions (i.e., documents) and stores them on the document suspense database.

If the user wishes to process the transaction immediately, the data entry/error correction program will perform an IMS message switch to an application edit/update program. If errors are detected, the application program will signal the data entry/error-correction program (2), which will post the errors highlighted back to the user, who may then correct the erroneous data and immediately resubmit the document.

Note that data can be entered and corrected without ever interacting with an application program. The data entry/error correction program also handles the scheduling of documents for processing. The purge-accepted-documents program (3) physically deletes documents from the document suspense database and creates an audit trail log.

Reference tables are created and maintained by the foundation software reference data edit/update software (4) and accessed through the reference data interface (RDI) software (5). The RDI is a memory-buffered approach, which takes maximum advantage of the fact that in most financial systems only a few specific table values constitute the majority of the requests. The RDI approach has eliminated over 90% of the

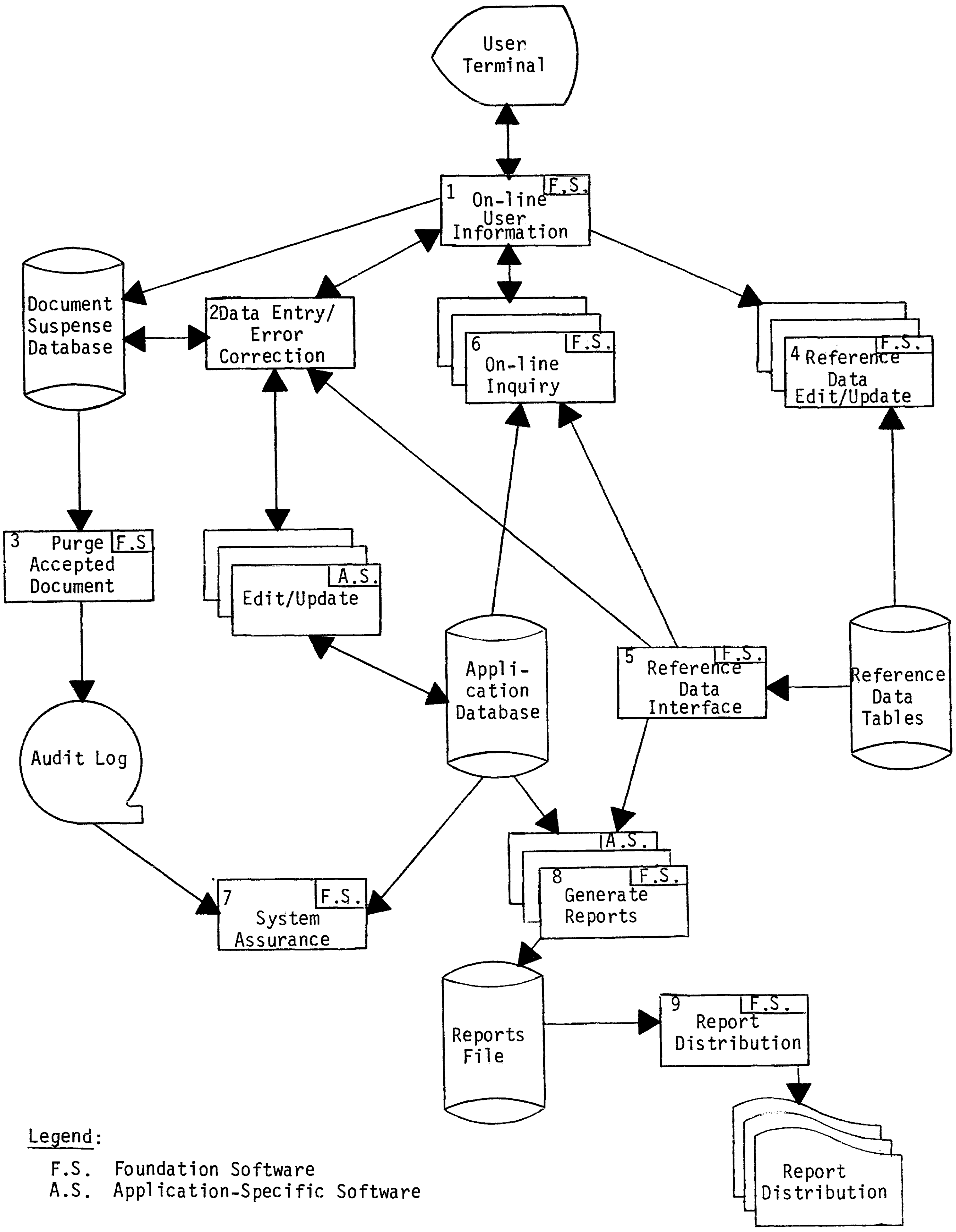

User Terminal
1 On-line User Information F.S.
Document Suspense Database
2 Data Entry/ Error Correction
6 On-line Inquiry F.S.
4 Reference Data Edit/Update F.S.
3 Purge Accepted Document F.S
Edit/Update A.S.
Application Database
5 Reference Data Interface F.S.
Reference Data Tables
Audit Log
7 System Assurance F.S.
A.S.
8 Generate Reports F.S.
Reports File
9 Report Distribution F.S.
Report Distribution
Legend:
F.S. Foundation Software
A.S. Application-Specific Software

Figure 3—IBM DB/DC integrated financial system architecture overview

reference data DL/I calls in our financial system for Standard Oil of Indiana.

The foundation software provides on-line inquiry (6) into the reference data tables and also provides ad hoc inquiry through a general-purpose inquiry package. The INQUIRY IV/IMS package from Informatics has been used for this purpose.

System assurance software (7) ensures that the application database retains integrity at all times. Not only is the *technical integrity* of the application database verified (no broken pointer chains, for example), but the *substantive integrity* is also verified. This capability is incorporated into the database design with planned redundancy and summary totals. In our experience this is an essential tool to help prevent system and application errors from corrupting the quality of the application data.

Reports are produced both by custom-written COBOL report programs, and by an ad hoc report generator (8), such as EASYTRIEVE/IMS from Panasophic Software, which is integrated into the foundation software.

Large systems typically generate scores of reports on a regular basis to be distributed to many recipients. It is time consuming and expensive to manually burst, duplicate, and decollate the output of standard report programs for distribution to individual managers and staff personnel. AMS has developed and used successfully on a wide variety of projects foundation software that generates a custom packet of report pages for each recipient. This report-distribution system (9) is table driven, and it allows each individual to receive the correct number of copies of the desired reports, all organized into a neatly bound and indexed packet.

A case for adaptable applications software

by MARY WOODWARD
Associates Financial Services
Southbend, Indiana
and
PETER F. DIGIAMMARINO
American Management Systems, Inc.
Redwood City, California

ABSTRACT

Contemporary economic circumstances have sent many organizations that extend consumer credit scrambling to secure automated support for collection operations. The traditional alternatives, custom system development and packaged software, fall far short of being acceptable to most large credit-oriented organizations. The Computer Assisted Collection System (CACS) was originally developed as a custom system and has since been used as adaptable foundation software by many large organizations to secure essentially customized support at a fraction of the cost, time, and risk that would normally be required. This paper reports on the success of the use of adaptive software to fill this urgent need and lends credence to the theory that throughout the 1980s there will be a trend towards the use of adaptive software to meet business' demands for low-risk, low-cost, fully functional and tailored software.

INTRODUCTION

Consumer credit privileges are among the most visible and popular services provided by financial institutions and stores. Recent government regulations, economic factors, and social trends have had a profound impact on the business of granting credit. Consequently, consumer credit operations are of paramount importance in many of today's business organizations.

A variety of support functions are required to establish and maintain a profitable consumer credit operation, including credit authorization, accounting, customer service, and collections. Changing economic conditions and restrictive legislation often strain an organization's capacity to provide effective and efficient credit services. The high payroll and record-keeping costs due to their labor-intensive nature also significantly affect productivity and profitability of credit operations. These factors have an especially severe impact on credit collections operations.

The Computer Assisted Collection System (CACS) is a software system that improves the productivity and effectiveness of credit collections through a form of office automation. CACS provides users with immediate, on-line access to pertinent account information to assist in executing collection tasks and in making decisions.

CACS was developed originally as a custom system by Wells Fargo Bank, N.A., in cooperation with American Management Systems, Inc. (AMS), a company that specializes in management consulting and computer systems development. The system first became operational in the Wells Fargo Credit Card Collection Department in the spring of 1980.

Wells Fargo, like many organizations, was severely affected by the recession of 1975. Collection operations were strained beyond capacity as the number of delinquent and overlimit customers requiring proper follow-up surged. Shortly after the recession had eased, Wells Fargo Bank resolved to develop an automated system for support of collections in order to lower processing costs, improve collection effectiveness, and accommodate surges in processing requirements in bad times, as well as to allow for aggressive growth in credit operations.

Efforts to develop an automated collection support system soon revealed that such a system would require careful human engineering, the application of design techniques not commonly found in contemporary automated support systems, and a technical architecture that, on the surface, appeared straightforward but, in reality, was quite complex. Several years and approximately one million dollars later, Wells Fargo was no closer to having an operational support system than at the outset.

A new, 18-month venture with AMS finally resulted in a comprehensive collection support system that, almost from the day of initial operation, started to pay back in terms of increased collection productivity (close to 100% increases in productivity were measured) and effectiveness (record low losses and delinquencies were experienced). Wells Fargo, had, after many hard years, succeeded.

THE CONCEPT OF ADAPTIVE SOFTWARE EMERGES

The systems developers thought that the underlying CACS design and technical architecture were sound enough and flexible enough to be used by other organizations to help them secure a collection system meeting their own collection requirements. CACS, while certainly not a traditional software package, could be used as baseline software around which a customized collection support system could be developed.

The merits of this logic were taken to be

1. Wells Fargo Bank's prior experience, and the experience of at least a half-dozen other organizations, indicated that it is difficult, expensive, time consuming, and risky to custom-develop a collection system.
2. There were no mainframe collection system packages available on the market.
3. Even if packages did emerge, the esoteric requirements found in large organizations, which have evolved over decades in the business, mean that the use of off-the-shelf software would require major business and operational concessions, which would be undesirable and traumatic.

For some set of organizations, CACS as foundation or adaptive software could, it was reasoned, be used to secure a significant *head start* towards developing a custom collection system. Based on available data, the development of a custom system for a large organization was estimated to require a budget of from $500,000 to $1.5 million for professional services over 18 to 36 elapsed months. With CACS, the same functionality could be accomplished for fees and services of from $100,000 to $500,000 in just 3 to 6 months.

On this reasoning, Wells Fargo and AMS set out to test their hypothesis in the market place. The target customers were large organizations in several market segments (banks, finance companies, service bureaus, retailers, etc.) who might be planning to provide automated support to collections.

THE INITIAL REACTION

Initial contact was made with several target corporations. As expected, many companies were making hasty plans to provide automated assistance to collections since the current recession was at hand and collection woes were mounting. While the need was well-established and immediate, the alternatives were not attractive. The options were to develop custom software or to conform to the terms offered by relatively inflexible, minicomputer-based collection systems that had recently emerged.

The concept of *starting with CACS* and building upon it was greeted initially by two types of response. Those organizations with large systems organizations tended to say that their company never bought packages because packages couldn't possibly meet their specific requirements, were not developed according to internal standards, and caused more trouble than they were worth. Organizations that would usually be inclined to consider off-the-shelf packaged software tended to focus on the fact that CACS was not actually a package at all because standard, well developed user documentation, run books, acceptance test scenarios, and the like did not exist.

The response to these points was that CACS represented a different kind of software solution. CACS is not a packaged system. It is a set of software that can be installed and built upon to provide, in the end, a complete system tailored to the organization's unique requirements. The system, in the end, belongs to the organization; it is unique and is maintained by in-house resources.

The decision to acquire and use CACS finally depends on a comparative analysis of functionality, cost, schedule, and risk (see Table 1). Without disputing any given organization's ability to succeed in developing a collection system, we see a good deal of empirical evidence that it is difficult to do so. The number of abortive efforts speaks for itself. For an organization starting from scratch, therefore, the risk of failure is high. The time and cost associated with custom development are also high relative to that required for systems that start with adaptive software. Finally, since the software is adaptive and easy to work with, all desired functions, features, and requirements can be accommodated.

The final analysis of the adaptive solution includes user review of the system in operation and technical evaluation of the system's components. The spectacular success of the system at Wells Fargo Bank, and later at other sites as well, left users clamoring for immediate installation of the system. The

TABLE I—Comparison of packaged software, custom software, and adaptive software

Evaluation Criterion	Packaged Software	Custom Software	Adaptive Software
Time required to implement/install	Low	High	Moderate to low
Costs to install	Low	High	Moderate to low
Risk	Low	High	Moderate to low
Degree of difficulty	Low	High	Moderate to low
Flexibility	Low	Variable	High
Closeness of fit to requirements	Low	High	High
Support	Rely on vendor	In-house	In-house

technical elegance of the system's underlying design methods and technical architecture left systems personnel convinced that the software would be easy to work with and maintain.

Though initial reactions to the concept of starting with CACS are often negative, the final decision is clearly in favor of the adaptive software because of its handling of the critical problems of development:

1. The need to reinvent the wheel is eliminated.
2. The organization's unique requirements can be met.
3. The risks are low.
4. The costs are a fraction of what they otherwise might be.
5. The time required is similarly reduced.

CACS STATE PROCESSING

There are several distinguishing design components of CACS that have primarily accounted for its success as adaptive software. The most prominent of these is its capacity for accurately and completely modeling work situations that conform to the following:

1. The work function is to manage a set of *items* according to prescribed procedures (e.g., clients, prospects, accounts, patients, etc.).
2. Associated with each item are one or more *events* that can occur (phone call, letter, payment, check-up, etc.).
3. For each item, one particular event is *scheduled* to occur next (e.g., follow-up telephone call on the tenth).
4. This next scheduled event defines the item's *state.*
5. It is essential to *track* all events.
6. Each event yields one of a set of possible *results* (e.g., no answer, insufficient payment, broken promise, bad credit check).
7. Given an item in a state, when an event occurs that yields a particular result, the prescribed procedures determine the *new state* for that item (e.g., after three no answers send a letter and obtain a credit report).

The item's progression through a well-defined set of work situations in response to the results of scheduled events can be documented using a standard state-processing diagram or a conventional finite-state automata grammar. Examples of this abound both in the world of computational theory[1] and in the world of collections. Figure 1 presents a simplified view of this from a collections viewpoint. Figure 2 presents an actual state diagram from one of the CACS installations. The system's *states* and processing rules are easily defined, set-up, used, and refined at a given installation (and between installations) using the CACS State Definition and Transition Tables, Parameter Tables, and well-structured application software. With this, the time required to understand, completely and unambiguously, and accurately document the business functions, and then to design in detail and implement automated support, is reduced to a mere fraction of that which would be required using a conventional approach.

The CACS State Processing facility, with the systems technical architecture (see Figure 3) and its other generic support

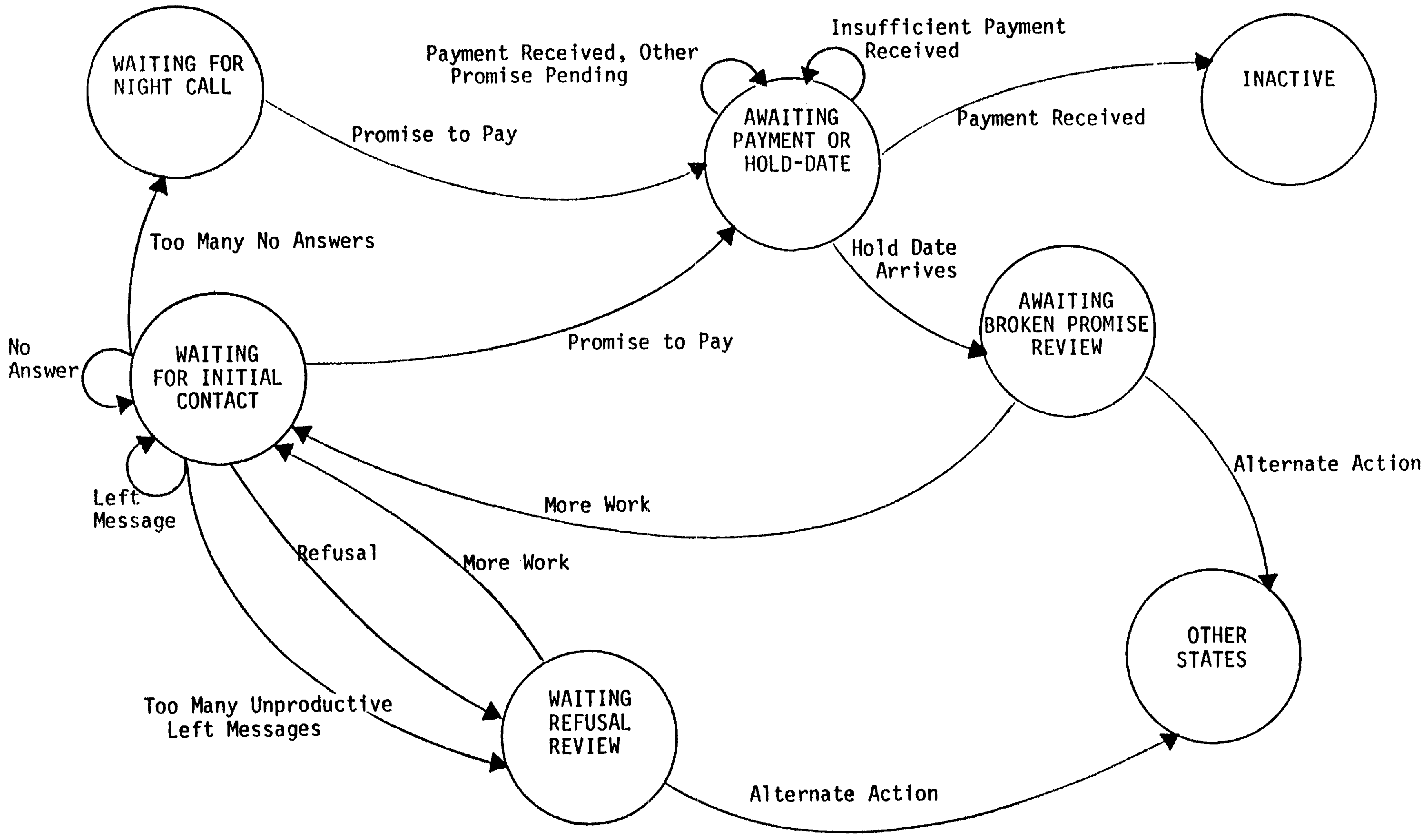


Figure 1—CACS state processor overview

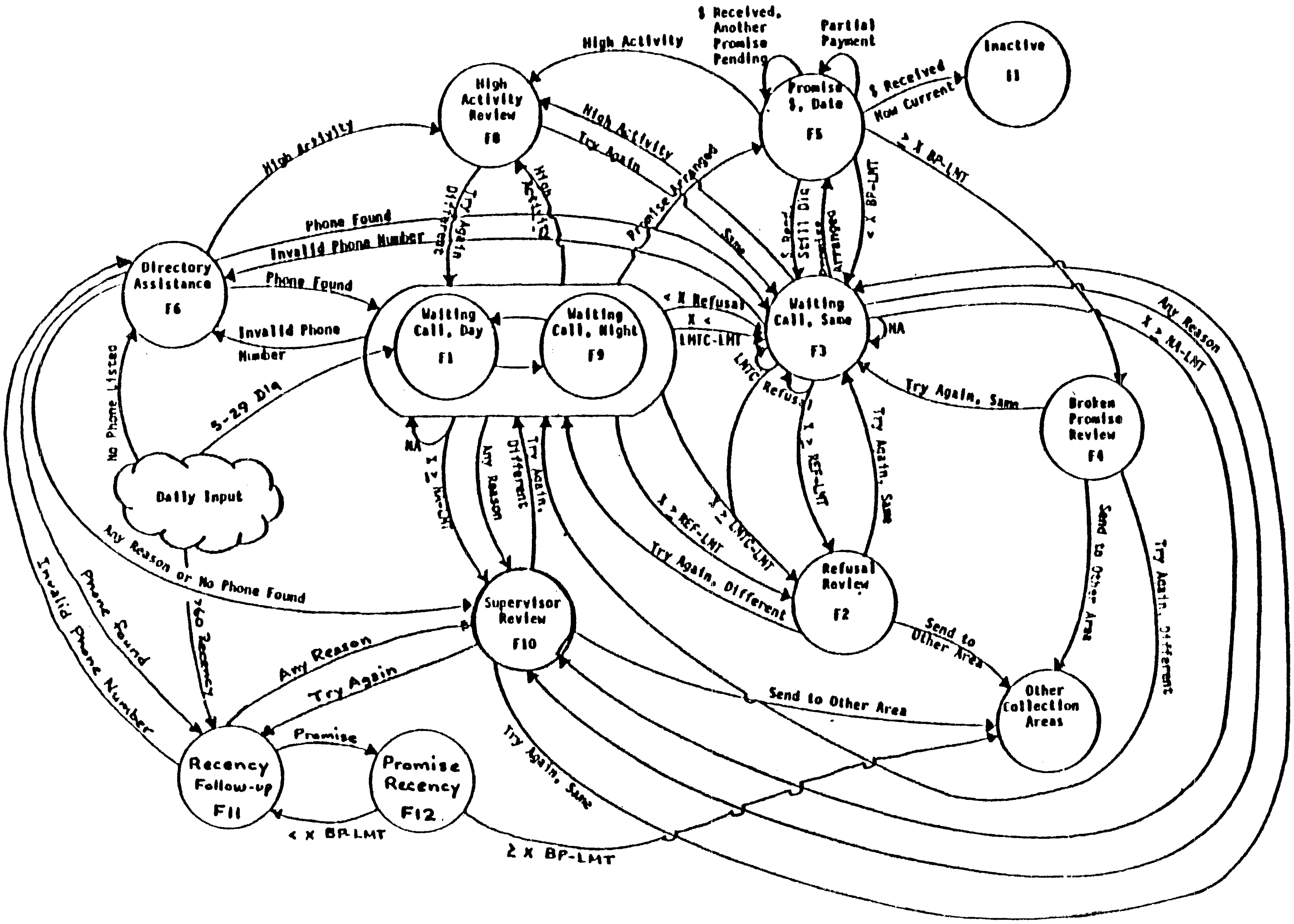


Figure 2—CACS state processor, diagram from actual installation

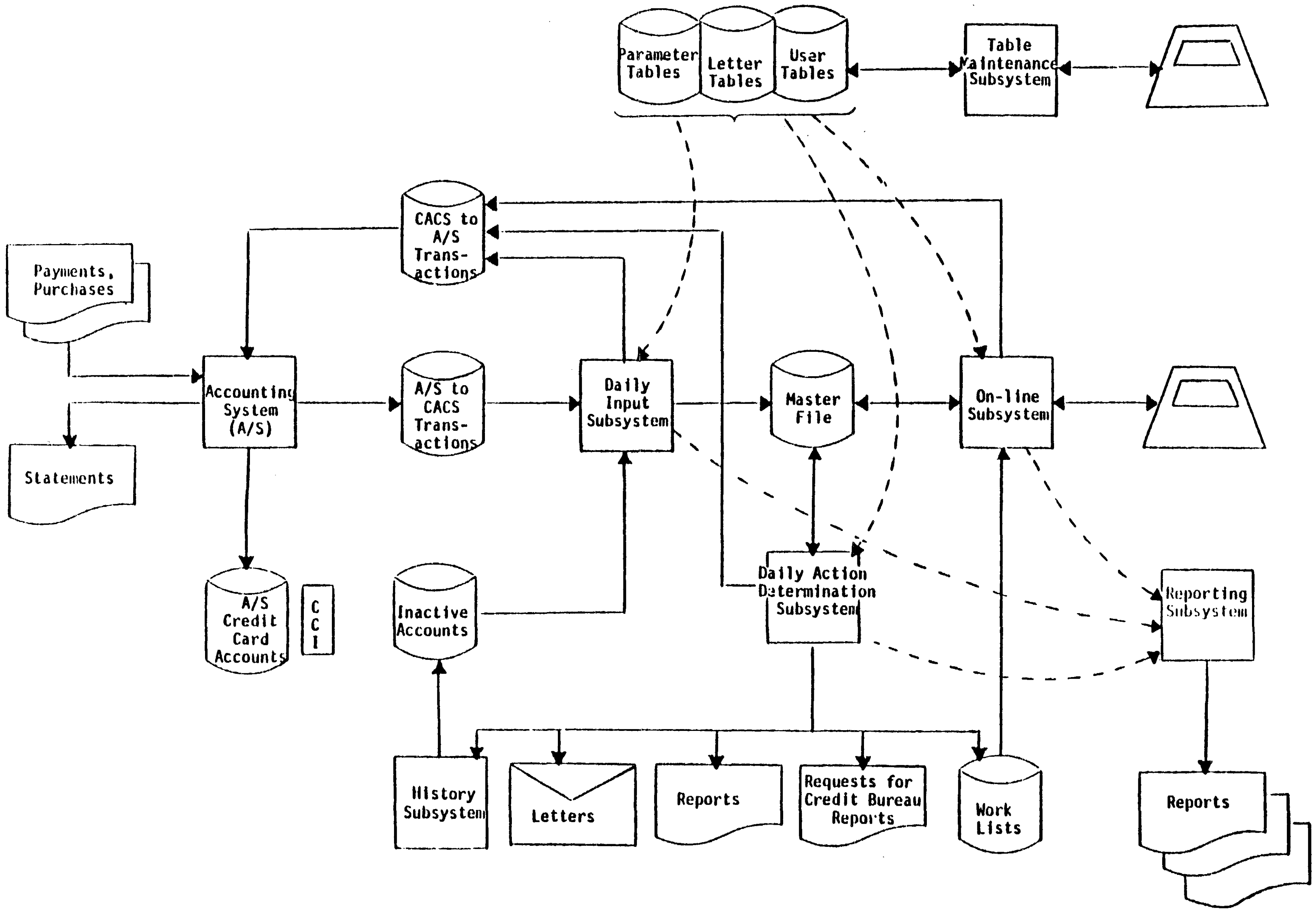


Figure 3—CACS technical architecture

functions (including list processing, audit trail processing, and historical data tracking), define a technology capable of being used as the foundation for a set of adaptive software systems.

IMPLICATIONS

Based on the CACS experience, there should be many similar success stories developed around adaptive software in the future. The need for multiple organizations to independently develop large and complicated custom systems to address the same fundamental business need is behind us. The need to modify the business functions to fit the packaged software solution is similarly obviated. The trend will be for well-designed software systems to be used as the foundation upon which other self-sustaining software solutions will evolve.

CACS is one of the forerunners in the evolution of such systems. Its success with this strategy has been both swift and startling. In just 14 months from the time of its introduction into the marketplace, 10 organizations have selected it for their solution. Half of these are already realizing the benefits from an operational collection system. The remainder will be operational within just a few months. Over 25 other organizations are now giving serious consideration to the system. The success of CACS as adaptive software is already secured. The success of adaptive software as a trend for the 80's lies just around the corner.

REFERENCES

1. Brainard, Walter, and Lawrence Landweber. *Theory of Computation.* New York: John Wiley, 1974.

Knowledgeable contexts for user interaction

by BOZENA HENISZ THOMPSON,
FREDERICK B. THOMPSON, and
TAI-PING HO
California Institute of Technology
Pasadena, California

ABSTRACT

ASK, A Simple Knowledgeable System, is a total system for the structuring, manipulation, and communication of information. The ASK user interface is a simple dialect of natural English. The system includes extensive means by which a user group and application programmer can build a knowledgeable context for user interaction. The users themselves can build, modify, and extend their knowledge base. They can add complex definitions that embody knowledge of their domain. They can ground a new tentative knowledge base on more stable ones, modifying and extending their new one without affecting the old.

A truly knowledgeable system must also know how to perform complex tasks in response to terse user inputs, taking over complicated but repetitive tasks on simple cues. The ASK system includes three system-guided dialogues that can be used to build such knowledgeability into a user's context.

INTRODUCTION

Systems for Experts

It is generally agreed that any computer system which directly serves a group of users must be knowledgeable concerning the domain in which that group is working. The term "knowledge base" is rapidly replacing "database" to describe the information available to the computer in responding to user interaction. One form of knowledge-based system that is receiving a good deal of attention is the *expert system.* In an expert system, experts build the knowledge base and users draw on this expert knowledge. In the words of Dr. Edward Feigenbaum, of Stanford University, whose seminal work established this important area:

> Expert systems can be viewed as intermediaries between experts, who interact with the systems in "knowledge acquisition" mode, and human users who interact with the systems in "consultation mode."

There are, on the other hand, many areas where the using group itself is intimately involved in the building, modification, and extension of their own knowledge base. In the typical research team, management or military staff, or business office, the central activity is the maintenance of the knowledge base in the form of plans, data, designs, and coordination of their operations. Office and manufacturing-automation systems will soon evolve into just such systems. Knowledge-based systems that support these activities must provide a kind of service to their users very different from the kinds provided by expert systems. They are, in the words of Dr. Donald Walker, of SRI International, Systems for Experts.

There are certain properties that a system for experts must have. First, such a system must be natural to use. This implies a reasonable facility for natural language but also for accepting the jargon that rapidly builds up within such a user-group. It implies a capability for text and graphic processing and for numerical and statistical calculation, all as an integral part of the knowledge base itself. Such systems must have means by which their user groups can easily add to, change, and extend their knowledge base as a normal part of their interaction with it. Such changes and extensions can come from many sources. Some will occur as part of the users' interaction with one another: bringing records up to date, writing reports and circulating comments on the work of others, completing design drawings, and scheduling and maintaining the information necessary for coordinating operations. Whole bodies of data may come from other sources, for example by the incorporation of commercially available data or the inclusion of the knowledge bases of subordinate groups. Providing facile means for effecting all of these varied kinds of modification and extension is an essential aspect of systems for experts.

When an expert team uses such a system on a day-to-day basis, many of their tasks will be repetitious, and with experience these tasks will reduce in essence to the specification of a few inputs to control a complex but repetitive process. A system for experts must provide the means for defining such tasks and relegating the tedious details to the system. And the system must know enough to hold a brief dialogue with its user to obtain the instructions necessary for it to complete the task. The specifications of such tasks will be an everyday practice of expert staffs; thus the means for such definition must be facile and a natural part of system interaction.

The ASK System, a System for Experts

This paper is about such a system, the ASK System, a system for experts. ASK, A Simple Knowledgeable System, exists. It is implemented on a Hewlett-Packard HP-9836 desktop computer, in PASCAL. The film shown in connection with this paper is of the ASK System in action. The film is taken in real time. (Many of the examples from the film are repeated in the written form of this paper.) In a single paper it would be impossible both to give a broad coverage and also to treat in any depth the various capabilities of this system. In this paper we have chosen the former to give a broad overview of the ASK System. For those who are seriously interested, we would be glad to find an appropriate opportunity to discuss more detailed aspects of the system.

To introduce you to ASK, we will start out with a few examples of queries of a simple database concerning ships. The uninitiated user may wish simply to ask:

```
〉 How many ships are there?
  8
〉 What is known about ships?
  some are in the following classes:
    Navy
    freighter
    old
    tanker
  all have the following attributes:
    destination
  some have the following attributes:
    home port
    cargo
  all have the following number attributes:
    age
  some have the following number attributes:
    speed
    length
    beam
〉 List the destinations and home port of each ship.
  ship           destination      home port
  Ubu            New York         Naples
                 Tokyo            —
  Maru           Oslo             Tokyo
  Kittyhawk      Naples           Boston
                 Boston           —
                 London           —
  Alamo          London           London
                 New York         —
  North Star     London           New York
  Nimitz         London           Norfolk
  Saratoga       unknown          Norfolk
〉 What is the number of New York ships?
  There are 2 answers:
    (1) New York (destination) ships
  2
    (2) New York (home port) ships
  1
〉 What about Norfolk?
  What is the number of Norfolk ships?
  Norfolk (home port) ships?
  2
〉 Wheat is carried by how many ships?
  2
〉 Coal and oil?
  coal and oil is carried by how many ships?
      ships
  coal    1
  oil     2
〉 How many ships are there with lnegth greater than 600 feet?
  Spelling correction: "lnegth" to "length"
  2
〉 What are they?
  What are they [ships with length greater than 600 feet]?
  North Star
  Kittyhawk
〉 What ships that carry wheat go to London or Oslo?
              ships that carry wheat
  London      Maru
  Oslo        Alamo
〉 Does the Maru carry wheat and go to London?
  yes
```

In the terminology of ASK, a user *context* is a knowledge base together with the vocabulary and definitions that go with it. A given user will usually have several contexts for various purposes, some of which may be shared with others in the group. You have seen examples from the small "ship" file context. Let's look at quite a different one, an example of a bibliography of literature on artificial intelligence (here truncated).

```
〉 How many articles are there about knowledge representation?
  15
〉 Which ones were written in 1979 by each author?
  Which [articles about knowledge representation] were written in 1979 by each author?
  author          articles about knowledge representation
  G. G. Hendrix   Encoding Knowledge in Partitioned
                  Networks
  R. J. Brachman  On the Epistemological Status of Se-
                  mantic Networks
〉 What other articles has Hendrix written?
  What other articles [than articles about knowledge
  representation] has Hendrix written?
  LIFER: A Natural Language Interface Facility
```

Or a knowledge base concerning one's mail:

```
〉 Who have I received mail from?
  John Smith
  Peter Jackson
〉 Display my mail from John Smith.
  Fred, Your budget looks good to me. Suggest you
  submit it as is.
    John
〉 File it in my budget correspondence file.
  File [my mail from John Smith] in my budget correspondence file.
  my mail from John Smith has been filed.
```

You have seen examples from three different contexts. How contexts are created and related to one another will be discussed below.

EXTENDING USER CONTEXTS

The major emphasis of this paper is on the ability of the users of ASK to structure their own knowledge bases. We turn now to those issues.

Extending the Knowledge Base and Adding and Changing Data

A knowledge base typically contains objects and classes of these objects; it also has knowledge of the attributes that these objects might possess and of relationships that might hold among them. The ASK System accepts objects of several types: individuals (e.g., Boston, John), numbers, texts (e.g., the text of a given memo or letter), matrices, and so on. The following sequence illustrates how a user can extend his or her knowledge base by adding new objects, classes, attributes, and relations. We will use the artificial-intelligence (AI) bibliography context for this illustration. Note how easy it is to add new attributes, "fields" in relational-database terminology, and how text objects are handled in a straightforward way as integral parts of the knowledge base.

```
⟩ Create the attribute: rating
  The attribute rating has been added.
⟩ create individuals: seminal, excellent, fair and impossible
  The following individuals have been added:
  seminal   excellent   fair   impossible
⟩ The rating of Winograd's 1980 paper in Cognitive Science is
  excellent.
  excellent has been added as the rating of Winograd's 1980
  paper in Cognitive Science.
⟩ rating of A Framework for Representing Knowledge is im-
  possible.
  impossible has been added as the rating of A Framework for
  Representing Knowledge.
⟩ Scripts, Plans, Goals and Understanding's rating is seminal.
  seminal has been added as the rating of Scripts, Plans,
  Goals and Understanding.
⟩ The rating of Minsky's paper in The Psychology of Com-
  puter Vision is seminal.
  impossible has been replaced by seminal as the rating of
  Minsky's paper in The Psychology of Computer Vision.
⟩ List the author of each paper that has a rating of either
  seminal or excellent.
  paper                                          author
  Scripts, Plans, Goals and Understanding   R. C. Schank
  A Framework for Representing Knowledge     M. Minsky
  What Does It Mean to Understand Natural
  Language                                   T. Winograd
⟩ Create an individual/text relation named comment
  The individual/text relation comment has been added.
⟩ Create a comment on A General Syntactic Processor
  Please enter text:
  Best paper on chart parsers.
  A new text has been added as comment on A General
  Syntactic Processor.
⟩ Display my comments on each paper by R. M. Kaplan.
  paper by R. M. Kaplan                      comments
  A General Syntactic Processor  Best paper on chart pars-
  ers.
```

Definitions

The examples so far illustrate how one can build, query, and modify a database. However, to make such a system more knowledgeable, one needs to be able to add definitions that embody interrelationships among the basic objects, classes and attributes of the data. The simplest form of definition is synonym:

```
⟩ definition: tub: old ship
  Defined.
```

Although this form of definition allows one to introduce abbreviations and many forms of jargon, more extensive forms of definition are desirable. Here are two illustrations using the ship file context. In the second definition, note the use of quotes to create local "variables."

```
⟩ definition: area: length * beam
  Defined.
⟩ List the length, beam and area of each tub.
  tub        length      beam      area
             foot        foot      foot**2
  Ubu        231.667     48        11120.016
  Alamo      564.5       84        47418.
⟩ definition: meter: 39.37 * (foot/12)
  Defined.
⟩ beam of the Alamo squared in square meters?
  655.526472343 square meters
⟩ definition: longest "ship": "ship" whose length is the max-
  imum length of "ship"s
  Defined.
⟩ What is the length in meters of the longest ship whose home
  port is Naples?
  121.920243840 meters
```

The notion of what is long may be quite different in another context, say in the bibliography context.

```
⟩ definition: long: paper whose number of pages exceeds 49
  Defined.
⟩ definition: long: book whose number of pages exceeds 800
  Defined.
⟩ What AI bibliography items are long?
  there are 2 answers:
  (1) long: paper whose number of pages exceeds 49
  Physical Symbol Systems
  A General Syntactic Processor
  (2) long: book whose number of pages exceeds 800
  Human Problem Solving
⟩ What long books were written in 1972?
  long: book whose number of pages exceeds 800
  Human Problem Solving
```

Family relationships make for a good illustration of definitions; we switch to a small family-relationships context.

```
⟩ What are attributes?
  individual/individual attributes:
  spouse
⟩ What are relations?
  individual/individual relations:
  parent
⟩ What are classes?
  individual classes:
  male
  female
⟩ What are definitions?
  definition: mother: female parent
  definition: father: male parent
  definition: child: converse of parent
  definition: sibling: child of parent but not oneself
  definition: cousin: child of sibling of parent
⟩ List the father and mother of each of Billy Smith's cousins.
  Billy Smith's cousins    father          mother
  Baby Boyd                Robert Boyd     Jill Boyd
```

Verbs

Initially, the only verbs known to the ASK System are "to be" and "to have." The user can add new verbs by paraphrase, as in the following illustration:

```
⟩ verb: ships "go" to New York: destination of ships is
  New York
  Defined.
⟩ verb: ships "carry" coal from London to Boston: ships have
  coal as cargo, have London as home port and go to Boston
  Defined.
⟩ Each old ship carries what cargo to each port?
  old ship     port        cargo
  Ubu          New York    oil
               Tokyo       oil
  Alamo        London      wheat
               ———         coal
⟩ What is carried by the Alamo?
  wheat
  coal
⟩ Wheat is carried to London from what ports?
  New York
⟩ What cities does the Alamo carry wheat to?
  London
```

Basing One Context on Another

We have shown how users can add new vocabulary, data, and definitions to their contexts. However, this would be a tedious way to build a large and useful database from scratch. We now discuss two ways of incorporating bodies of existing data in a user context.

Consider a user of the AI bibliography context illustrated above, who wants to build a wider bibliography context, adding new information—vocabulary, data, and definitions—without, however, disturbing the old one. To do so, all he or she needs to do is select a new name, say CS bibliography, and type

Base CS Bibliography on AI Bibliography

The result of this basing action is a new context. Upon entering this new context—

⟩ Enter CS Bibliography

—one can make additions:

```
⟩ individuals: An Introduction to Database Systems, C. J.
  Date
  The following individuals have been added:
  An Introduction to Database Systems C. J. Date
⟩ An Introduction to Database Systems is a book.
  An Introduction to Database Systems has been added to
  book.
⟩ The author of An Introduction to Database Systems is C. J.
  Date.
  C. J. Date has been added as author of An Introduction to
  Database Systems.
⟩ Keyword of An Introduction to Database Systems is data-
  base.
  database has been added as keyword of An Introduction to
  Database Systems.
⟩ Who wrote what about databases?
  author
  D. L. Waltz  Natural Language Access to a Large Data Base
  C. J. Date   An Introduction to Database Systems
```

These additions to the CS bibliography would not affect the AI bibliography context. However, additions and modifications that are subsequently made in the AI bibliography context would automatically be reflected in the CS bibliography. Several contexts can be based on a given one, and one context can be based on several; thus a hierarchical structure of contexts can be realized. All contexts are directly or indirectly based on the BASE context, which contains the function words and grammar of the ASK dialect of English, the mathematical and statistical capabilities, and the word processor.

The Bulk Data Input Dialogue

There is a great deal of information in existing databases, and a system for experts must facilitate the addition of such data to the knowledgeable user's context. In the ASK System there is a dialogue, called the Bulk Data Input Dialogue, which can be called on to build an existing database into one's context. The result not only integrates these new data with those already in the context, according to the ASK dialect of English, but in many circumstances will make the use of these data more responsive to users' needs.

The Bulk Data Input Dialogue prompts the user for necessary information to (a) establish the physical structure of the database to be included (b) add necessary classes and attributes as needed for the new data entries. The user also indicates, using English constructions, the informational relationships among the fields in the physical records of the database file that he or she wishes carried over to the ASK context. We will not illustrate the Bulk Data Input Dialogue here, since it is similar to two other ASK System dialogues that will be described and illustrated below.

KNOWLEDGEABLE DIALOGUES

In the day-to-day use of an interactive system, a user is very often involved in repetitive tasks; much of the drudgery of such tasks could be shifted onto the system if it were more knowledgeable. Such a knowledgeable system, as it goes about a task for the user, may need additional information from the user. What information it needs at a particular point may depend on earlier user inputs and the current state of the database.

Some have raised the question, whether natural language is always the most desirable medium for a user's communication with the computer. Expert systems, for example, have tended to use computer-guided dialogues. One simple form such a dialogue might take is illustrated by the following dialogue, in which a new entry is added to the AI bibliography:

```
⟩ New bibliography item
⟩ Add to what bibliography? AI Bibliography
⟩ Title: Natural Language Processing
⟩ Author: Harry Tennant
⟩ Keyword: natural language
⟩ Keyword: syntax processing
⟩ Keyword: speech acts
⟩ Keyword:
  Natural Language Processing has been added to the AI
  Bibliography.
⟩ Title:
  The "new bibliography item" dialogue is completed.
⟩ What AI Bibliography items were written by Harry Ten-
  nant?
  Experience with the Evaluation of Natural Language
  Question Answerers
  Natural Language Processing
```

Other alternative media for user/system communication are menu boards, selection arrays, and query by example. Many other cryptic ways to communicate user needs to a knowledgeable system can be thought of; often the most useful means will be highly specific to the application. For example: in positioning cargo in the hold of a ship, one would like to be able to display the particular cargo space, showing its current cargo, and to call for and move into place other items that are to be included.

In the past, enabling the system to respond more intelligently to the user's needs required the provision of elaborate programs, since the user's tasks may be quite involved, with complex decision structures. The introduction of terse, effective communication has incurred long delays; thus a user's changing needs had little chance of being met. In the ASK System, the users themselves can provide this knowledge. They can tell the system how to elicit the necessary information and how to complete the required task. This ASK capability is quite easy to use, opening the way for its everyday use in extending the knowledgeable responsiveness of the computer to the user's immediate needs.

The Dialogue-Designing Dialogue

The user must provide the system with knowledge of a particular task; more precisely he or she must program this knowledge into the system. The result of this programming will be a system-guided dialogue that the user can subsequently initiate and that will then elicit the necessary inputs that it needs. Using these inputs in conjunction with the knowledge already available, particularly the database, the system completes the task. It is this system-guided dialogue that the user must be able to design.

In the ASK System, there is a special dialogue that can be used to design system-guided dialogues to accomplish particular tasks. We call this the Dialogue Designing Dialogue (DDD). Using DDD, the user becomes a computer-aided designer. Since DDD, in conducting its dialogue with the user, only requires simple responses or responses phrased in ASK English, the user need not have any programming skill or experience at all. Using DDD, the user alone can replace a tedious, repetitive task with an efficient system-guided dialogue, all in a natural-language environment. The ASK DDD constitutes a high-level, natural-language programming capability. We hasten to add that it is not a general-purpose programming environment. It is for "ultra-high"-level programming, gaining its programming efficiency through the assumption of an extensive vocabulary and knowledge base on which it can draw.

DDD is based on the concept of an *interaction node.* Such a node represents a point in the dialogue where the computer turns to the user for additional input, that is, more data or further instruction as to what is desired. At such a node, the system prompts the user as to what information it needs, digests the user's response, takes indicated actions, and progresses to another node that it perceives as the next place for interaction. As it does so, it maintains its own local context, remembering what the user has told it and what it is supposed to do.

The DDD dialogue sets up all of this; therefore it is itself quite complex in its dialogue paths to elicit the information it needs, information about prompts, expectations of user responses, diagnostic messages, references to the database, maintenance of the local context, and so on. We can only illustrate a small part of the DDD dialogue here, but it should give you some feel for how the DDD dialogue works. Earlier in this paper, we used as an illustration a simple dialogue for adding a new item to the AI bibliography. Here is the DDD dialogue that defined that "new bibliography item" dialogue.

```
⟩ new dialogue
⟩ What user input should initiate this dialogue? new bibli-
  ography item
  Please define each node in turn.
  Designing node 1:
⟩ What is the prompt message for node 1: Add to what bibli-
  ography?
⟩ If you wish the response to this prompt to be assigned to a
  field, give the field number: 1
⟩ What is the type option for node 1: 2
⟩ What is the response option for a nil response to node 1: 1
⟩ What is the response option for a bad evaluation of user
  response: 2
⟩ What is the response option for a good evaluation of user
  response: 2
⟩ Transfer to what node: 2
  Node 1 is completed.
  Designing node 2:
  The fields that have been assigned on entering this node
  are:
  <1>: Add to what bibliography?
⟩ What is the prompt message for node 2: Title:
⟩ If you wish the response to this prompt to be assigned to a
  field, give the field number: 2
⟩ What is the type option for node 2: 2
⟩ What is the new word option: 3
⟩ What is the category option for this word: 1
⟩ What is the response option for a nil response to node 1:
  help
  1: abort the dialogue and exit
  2: reprompt
  3: complete all actions and exit
  4: a condition—action—transition sequence
  5: transfer to next node directly
⟩ What is the response option for a nil response to node 2: 3
⟩ What is the response option for a bad evaluation of user
  response: 2
⟩ What is the response option for a good evaluation of user
  response: 4
⟩ Condition: otherwise
⟩ Action: <2> is a <1>.
⟩ Action:
⟩ Transfer to what node: 3
  Node 2 is completed.
  Designing node 3:
  The fields that have been assigned on entering this node
  are:
  <1>: Add to what bibliography?
  <2>: Title:
⟩ What is the prompt message for node 3: Author:
⟩ If you wish the response to this prompt to be assigned to a
  field, give the field number: 3
⟩ What is the type option for node 3: 2
⟩ What is the new word option: 3
⟩ What is the category option for this word: 1
⟩ What is the response option for a nil response to node 3: 2
⟩ What is the response option for a bad evaluation of user
  response: 2
⟩ What is the response option for a good evaluation of user
  response: 4
⟩ Condition: otherwise
⟩ Action: Author of <2> is <3>.
⟩ Action:
⟩ Transfer to what node: 4
  Node 3 is completed.
  Designing node 4:
  The fields that have been assigned on entering this node
  are:
  <1>: Add to what bibliography?
  <2>: Title:
  <3>: Author:
⟩ What is the prompt message for node 4: Keyword:
⟩ If you wish the response to this prompt to be assigned to a
  field, give the field number: 4
⟩ What is the type option for node 4: 2
⟩ What is the new word option: 3
⟩ What is the category option for this word: 1
⟩ What is the response option for a nil response to node 4: 4
⟩ Condition: otherwise
⟩ Action: display: "<2> has been added to the <1>."
⟩ Action:
⟩ Transfer to what node: 2
⟩ What is the response option for a bad evaluation of user
  response: 2
⟩ What is the response option for a good evaluation of user
  response: 4
⟩ Condition: otherwise
⟩ Action: Keyword of <2> is <4>.
⟩ Action:
⟩ Transfer to what node: 4
  Node 4 is completed.
  Design of "new bibliography item" dialogue is completed.
```

Simple dialogues, like this one, can be defined by the user. However, complex decision/action structures and the provision of diagnostics and recovery, complex looping, and so forth is the appropriate province of the application programmer. The "node-driven" organization of DDD is quite natural for someone with even brief experience in computer programming. Sketching the dialogue as a rough flow chart, then proceeding to the use of DDD, one can quickly implement complex processes. DDD has a number of features that facilitate the development of the program, including a variety of validity checks. The super-high level of natural-language programming means that the sort of bugs found at low levels are eliminated. Particularly significant is the fact that in the development of such a user dialogue, all of the vocabulary and associated semantics of the immediate application are directly available.

The Use of Forms as a Dialogue Medium

The *form* is an efficient means of communication with which we are all familiar. A number of computer systems include a forms package. For most of these, however, filling in a form results only in a document; the form does not constitute a medium for interacting with the knowledge base or controlling the actions of the system. The ASK forms capability enlarges the roles and ways in which forms can be used as a medium for user interaction. As the user fills in the fields of a form, the system can make use of the information being supplied to (a) check its consistency with the data already in the knowledge base and, if necessary, respond with a diagnostic, (b) fill in other fields with data developed from the knowledge base, (c) extend the knowledge base, adding to the vocabulary and adding to or changing the data itself, and (4) file the completed form in prescribed files or in those indicated by the user and also mail it to a specified distribution list through the electronic-mail subsystem. Since the form processing can check consistency and modify the knowledge base, forms can be used to facilitate data input. Since form processing can fill fields in the form, the forms capability includes the functions of a report generator. Letters and memos can be written as special cases of form filling, automatically adding dates, addresses, and so on, and filing and dispatching the result.

It must be easy and natural to add new forms, if they are to be a convenient tool. That is the function of the Forms Designing Dialogue. Much like the Bulk Data Input Dialogue and the DDD, the Forms Designing Dialogue holds a dialogue with the user through which he or she can specify the fields of the form itself and the processing to be automatically accomplished at the time the Form is filled in. The Forms

Designing Dialogue makes many options available, only a few of which are illustrated here.

The protocol to be included here as an example of the Forms Designing Dialogue is designed to illustrate the features mentioned above, and does not reflect any actual form. It utilizes the notion of a field as in the DDD, allowing a field to be referred to by number.

```
⟩ new form
⟩ form name: shipping
⟩ start your design:

                    Shipping Form
ship: 2222222222222222
port: 2222222222222222

quantity  item                  price    total
1111      2222222222222222222   $111.11  $1111.11

commander: 2222222222222222

                    Shipping Form
ship: <1>2222222222
port: <2>2222222222

quantity  item                    price       total
<3>1   <4>22222222222222          $<5>.11     $<6>1.11
commander: <7>2222222222

⟩ Please confirm the design ("y" or "n"): y
⟩ Do you want to repeat some lines? (y/n): y
⟩ repetition field: <3>
⟩ repetition frequency: 0
⟩ repetition field:
  What fields are to be filled by the computer?
⟩ field: <2>
⟩ value: home port of <1>
⟩ if unable, what message should be used:
  <1> does not have a home port.
⟩ if unable, what option should be taken: 1
⟩ field: <5>
⟩ value: price of <4>
⟩ if unable, what message should be used:
⟩ if unable, what option should be taken:
⟩ field: <6>
⟩ value: <3> * <5>
⟩ if unable, what message should be used:
⟩ if unable, what option should be taken:
⟩ field:
  What conditions should be verified by the computer?
⟩ condition: <3> is less than the inventory of <4>?
⟩ if fail, what message should be used?
  There are not enough <4>s in stock.
⟩ if fail, what option should be taken? 2
⟩ condition:
  Actions to be taken on completion of the form?
⟩ update data base: Reduce inventory of <4> by <3>.
⟩ update data base: Commander of <1> is <7>.
⟩ update data base:
⟩ name of completed form: Shipping List for <1>
⟩ file in: Shipping Invoice File.
⟩ file in:
⟩ mail to: Jones.
⟩ mail to: ask
  The design of form "shipping" has been completed.
```

We will now fill in the form that was just designed. To illustrate the fact that the database will be changed as a result of filling the form, we first ask the questions:

```
⟩ What is the home port and commander of each old ship?
  There are 2 answers:
  (1) There is no commander.
  (2)
  ship       home port
  Ubu        Naples
  Alamo      London
⟩ Who is John Smith?
  The following words are not in the vocabulary: John Smith
⟩ Inventory of wheat and hydrochloric acid?
  wheat and hydrochloric acid              inventory
  wheat                                         86.7
  hydrochloric acid                          123400.
```

Note that the home port of the Alamo is London and that it does not have a commander, further that John Smith is not known to the system.

```
⟩ Fill shipping
```

(For the purposes of the published paper, in contrast to the film shown at the presentation of the paper, only the initial and final copies of the form are given, underscores indicate fields filled in by the user, all other fields being automatically filled by the System)

(before)

```
                    Shipping Form
ship: _
port:

quantity    item                    price     total
                                    $   .     $   .
commander:
```

(after)

```
                    Shipping Form
ship: Alamo
port: London

quantity    item                    price      total
3           wheat                   $ 35.75    $ 107.25
500         hydrochloric acid       $  2.50    $1250.00

commander: John Smith
```

Shipping List for Alamo has been filed in Shipping Invoice File.

```
Shipping List for Alamo has been mailed to Jones.
mail to:
  Fill shipping has been completed.
〉 List the home port and commander of each ship.
  ship      home port    commander
  Ubu       Naples       ———
  Alamo     London       John Smith
〉 Inventory of wheat and hydrochloric acid?
  wheat and hydrochloric acid          inventory
  wheat                                     83.7
  hydrochloric acid                      122900.
〉 What is in the Shipping Invoice File?
  Shipping List for Alamo
  Shipping List for Maru
```

ACKNOWLEDGMENTS

The three system-guided dialogues, Bulk Data Input, Dialogue Designing Dialogue, and Forms Designing Dialogue, are from the doctoral dissertation of Tai-Ping Ho. The aspects of ASK that concern the basing of one context on another are from the doctoral dissertation of Kwang-I Yu. The methods for handling anaphora, fragments, and correction of inputs are from the doctoral dissertation of David Trawick.

The ASK System is implemented on the HP-9836 Desktop Computer in PASCAL. The research that has resulted in the ASK System has been supported by the Hewlett-Packard Corporation.

An English-language processing system that "learns" about new domains

by BRUCE W. BALLARD and JOHN C. LUSTH
Duke University
Durham, North Carolina

ABSTRACT

We are developing an English-language processing system called LDC with emphasis upon (a) small- or medium-sized *office* domains, as opposed to large relational-style databases; (b) mechanisms to *learn* about new domains and the English to be used in discussing them; and (c) capabilities for *deep* semantic processing, for example where English inputs can be phrased naturally, not merely as a notational variant for complete, formal queries. LDC consists of two major components and an external retrieval module. The first component, which we call "Prep," obtains information about a new domain and the language to be used in discussing it. The second, "user-phase," component of LDC resembles an ordinary NL processor, but (a) most decisions are determined from the preprocessed information appearing in the data files produced by Prep, and (b) the emphasis is upon the semantics of "layered" domains, described herein. In this paper we (1) present the motivation behind LDC; (2) summarize and give examples of the behavior of Prep; (3) provide an overview of the user-phase component; and (4) give examples of current and projected capabilities of the system.

INTRODUCTION

During the 1970's, several experimental natural-language processing systems were developed, many of them reaching the prototype stage. At least one natural-language database query system is now being marketed,[10] while several other systems have been successfully used in pilot studies. Although most practical investigations have concerned database query,[6,7,10,13,15–17,19,21] our previous work in natural-language processing at Duke has been in the area of natural-language programming. Our system, called the Natural Language Computer (NLC), was developed in the late 1970's,[2,4] has been systematically evaluated,[5] and has been used on a trial basis by linear algebra students without prior computing experience.[8]

Drawing on our experience with NLC, we are presently developing a new NL processor, which we call the Layered Domain Class (LDC) system. Our current emphasis is upon (a) small- or medium-sized *office* domains, as opposed to large relational-style databases, (b) mechanisms to *learn* about new domains and the English to be used in discussing them, and (c) capabilities for *deep* semantic processing, e.g., where English inputs can be phrased naturally, not merely as a notational variant for complete, formal queries. Our approach amounts to constructing a "knowledge base" and is closer in spirit to systems like KLAUS, POL, SCHED, and TEAM than to the database systems cited above.[9,12,14,18]

In a recent publication[1] we presented the philosophical and psychological basis on which LDC is founded. As is explained there, we are currently considering the class of what we refer to as "layered" domains. We regard a domain as layered if its structural relationships resemble those of NLC matrices, where entities break down uniformly into lower-level entities. Examples of "layered" domains and their associated entity breakdowns include *matrices* (matrix, row/column, entry); *desk calendars* (year, month, week, day, hour slot); *office architecture,* (building, floor/wing, room); *document organization* (document, section, paragraph, sentence, word, character); and *academic course offerings* (course, section, student). While some of these domains (e.g., academic course enrollments) can be represented fairly well by conventional database schemes, and others (e.g., documents) as text files, we believe experience with previous systems has proven that many of the semantics arising in a natural-language environment require a "deeper" form of domain model than are provided by conventional representations. This is why LDC has chosen to formulate a model of "layered" domains and seeks to work with them. In this paper, we will select examples from the course offering and document domains to illustrate the capabilities and goals of LDC.

LDC consists of two major components and an external retrieval module. The first component, which we call "Prep," serves to acquire information about a new domain and the language to be used in discussing it. It then uses this domain knowledge to derive various forms of information that will be used during subsequent processing. The second, "user-phase" component of LDC resembles an ordinary NL processor, but its design is highly parameterized. Thus, many of its decisions are determined from the preprocessed information appearing in the data files produced by Prep. The actual retrieval component and raw-data file are regarded as external to LDC.

THE LEARNING COMPONENT

The initial interaction between a user and LDC, which involves telling the system about a new domain, consists of a dialogue with the preprocessor, which we call "Prep." Prep operates by asking for (a) the names of each type of "entity" of the domain, (b) the nature of the relationships among them, (c) the English words that will be used as nouns, verbs, and modifiers, and (d) morphological and semantic properties of these new words. It also allows the user to probe its knowledge and to make updates as desired. We will now summarize the capabilities of Prep.

Domain Structure Acquisition

Suppose we want to tell Prep about a data file that records all student grades in a certain academic department for a given semester. By asking the user a series of questions, Prep constructs a "domain model" network like that shown in Figure 1. The important distinctions here are (a) *decomposition* of one entity into another, indicated by the double arrow, versus a simpler form of association; (b) *multiple* values, indicated by plural names, versus single values; (c) an idea of which multiple values are to be thought of as *ordered,* indicated by the asterisk, for which ordinals such as "first," "second," ..., "last" may be applied; and (d) an indication of which nodes will have *persons* as values, indicated by an exclamation point. The actual internal network is represented in a nested list format. A description of how this structure is acquired, including actual dialogue with the user, can be found elsewhere.[1]

Language Acquisition

Having learned about domain structure, Prep proceeds to inquire about related language items. In particular, for each entity of the domain (node of the domain-structure network),

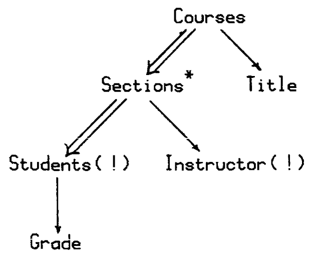


Figure 1—An internal model of the final grades domain

Prep asks for (a) nouns used to refer to the entity, (b) adjectives which modify the entity, and (c) nouns used to modify the entity. Prep then proceeds to ask for verbs having the given entity as subject. Since our parser and semantics processor work with slotted, case-frame structures, Prep asks for the entity types that are allowed for subject, object, and as prepositional arguments. For instance, Prep might be told that an *instructor* can *fail* a *student,* that a *student* can *fail* a *course,* that a *student* can *take* a *course from* an *instructor,* and so forth.

Having acquired a list of new vocabulary items, Prep proposes what its rules suggest will be their inflections, for example past and present participles for verbs, comparative and superlative forms for adjectives, plurals for nouns, and so forth. The user may then either accept the system's "guess" or specify the correct form.

Finally, Prep asks about the *meanings* of adjectives and verbs, which it attempts to capture in terms of seven primitives. This small set, coupled with mechanisms for traversing the domain-structure network, provides a powerful language with which to describe verb and adjective meanings. The seven primitive functions are *id,* the identity function; *val,* which returns the "value" or name field of a record; *look,* which retrieves a specified field of a record; *num,* which returns the size of its argument, which is assumed to be a set; *sum,* which returns the sum of its list of inputs; *avg,* which returns the average of its list of inputs; and *pct,* which returns the fraction of its list of boolean arguments which are true. In addition to these seven built-in functions, other user-defined adjectives can also be used. Thus, a "wordy" manuscript might be described as a manuscript where 60% or more of the sentences are "long." The specification of these two adjectives is shown in Figure 2.

Since verbs and adjectives are treated similarly, we will confine our discussion to adjectives. Furthermore, once the meaning of an adjective has been obtained, the semantics for its associated comparative and superlative forms can be derived automatically. For example, the definition of "long sentence" as given in Figure 2 says to (a) compute a *value* for the entity in question, in this case the number of words in the sentence, and then (b) *compare* this value against a designated *parameter,* in this case 20. When phrases such as "sentences longer than . . . " and "the longest sentence" arise, values are computed as given in step (a) but then compared against one another, rather than against the specified parameter.

```
ACQUIRING SEMANTICS FOR LONG SENTENCE
PRIMARY?     sentence
TARGET?      word
PATH IS:     WORD/SENTENCE
FUNCTIONS?   id  /num
PREDICATE?   >= 20

ACQUIRING SEMANTICS FOR WORDY MANUSCRIPT
PRIMARY?     manuscript
TARGET?      sentence
PATH IS:     SENTENCE/PARAGRAPH/CHAPTER/MANUSCRIPT
FUNCTIONS?   long     /id       /id      /pct
PREDICATE?   >= 60
```

Figure 2—Semantic acquisition for adjectives (system out in upper case)

As shown in Figure 2, Prep requests four pieces of information for each adjective-entity pair. The first of these, called the *primary,* is the entity to which the adjective is actually "applied." This may or may not be the entity with which the adjective has been associated. For example, if a "lucky" student is one who is enrolled in the section of a good instructor, instructor would be the primary, not student, since the instructor would have to be investigated.

The second piece of information needed, called the *target,* is any descendant of the primary and is the most primitive entity needed to specify the semantics. In the case of "long sentence," the target entity would be word, since the number of words must be found to tell whether a sentence is long.

The third piece of information requested is a *list of functions* corresponding to the arcs between the primary and the target nodes. Once the primary and target are identified, Prep outputs a path from the target to the primary. This path serves as a guide to the user in specifying what is to be done to each piece of data as it bubbles its way upward during semantic processing.

The fourth and final piece of information required is a *predicate* to be applied to the numerical value obtained from the series of function calls just acquired. The overall effect, then, of applying an adjective to a datarep is to obtain either "true" or "false."

Probing and Update

At any time during acquisition, Prep allows the user to review all or selected parts of the information associated with a given term or syntactic class. For instance, the user can ask Prep to display all entity types presently known—

```
ENTITY NAME? list
ENTITIES ARE: SECTION, GRADE, STUDENT
```

—or may ask for information about all ways in which specified terms can be used—

```
ENTITY NAME? uses score poor
score: SYNONYM FOR GRADE,
       VERB FOR STUDENT
poor:  ADJECTIVE FOR STUDENT.
```

Typing "uses" without arguments instructs Prep to give information on all known terms.

When the user decides to change previous specifications, he or she may call for an update, which begins with Prep asking which entity is to be updated.

ENTITY NAME? instructor

Next, a menu of update options is displayed.

WHAT INFORMATION NEEDS TO BE UPDATED FOR INSTRUCTOR?

LIST ATTRIBUTES	(TYPE list)
DELETE ENTITY	(TYPE del)
RENAME ENTITY	(TYPE ren)
SYNONYMS	(TYPE syn)
TYPE	(TYPE typ)
. . .	

The *list* option is used to request Prep to summarize all information associated with the entity in question. The options *delete* and *rename* are provided to allow for correcting errors or for making changes in the network structure. The remaining options are listed in the order the associated information was originally acquired. A verb specification may be altered by updating the entity associated with its subject.

As an example of an update, suppose the user has neglected to inform Prep that the word "teacher" can be used in place of "instructor." This correction is accomplished as follows:

OPTION: syn
SYNONYMS FOR INSTRUCTOR ARE: PROFESSOR
SYNONYMS TO BE ADDED: teacher
SYNONYMS TO BE DELETED: none

Here we see that Prep has (a) listed current synonyms; (b) asked for new ones; and finally (c) asked for old ones to be removed.

USER-PHASE PROCESSING

As shown in Figure 3, the user-phase component of LDC is designed as a linear stage of modules for scanning, parsing, semantic processing, and output generation.

Scanning

The role of the *scanner* is to identify each word of the typed or spoken input and retrieve information about it from the dictionary file, which will have been created by Prep. Each dictionary listing consists of (a) the *word* itself; (b) its *part of speech*; (c) the associated *root* word; and (d) zero or more associated *features,* each with one or more possible values. An example of a word definition might be

(better Compar good (nt student section))

which says that "better" is a comparative form of "good," and can be applied to nouns having an nt-feature (for "nountype") of either "student" or "section." Some words will have more than one dictionary listing, in which case the scanner sends

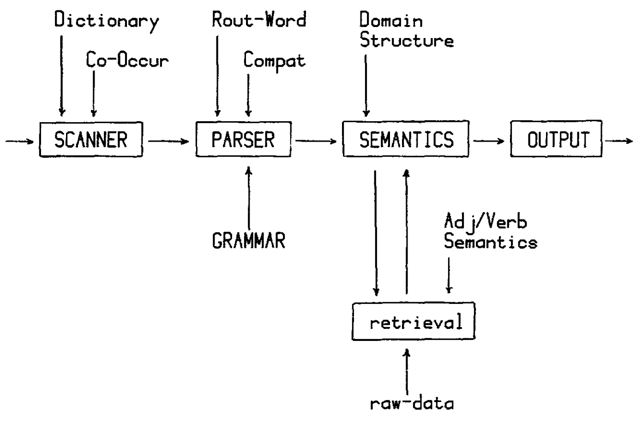


Figure 3—An overview of user-phase processing. (Names of program modules in boxes; other names are names of files, all of which are created by the Preprocessor except for the grammar and the raw-data file. The retrieval module is regarded as external to LDC.)

them all to the parser, where context will be used to select one of them.

The existing LDC scanner assumes typed input. However, we have been experimenting with a Nippon DP-200 voice recognition device, have completed an initial interface between it and NLC, and will eventually want to provide the LDC user with a microphone rather than a keyboard. One of the structures we are using tells which pairs of dictionary words can occur next to each other in a legal input. Anticipating the introduction of voice technology into LDC, Prep automatically constructs this word-pair information by consulting (a) the system grammar and (b) the domain-specific dictionary it is creating. This file has been called Co-Occur in Figure 3.

Parsing

The job of the *parser* is to determine, from the information provided by the scanner, the syntactic *structure* of the input. In a computational domain, especially one for retrieval rather than programming, the syntactic complexity of most inputs lies in the complexity of their noun phrases. For this reason, we regard relative-clause verb forms as basic, and sentence-level verbs as derived. For example, the input

"How many students failed the midterm?"

is treated as

Find-size-of: "students who failed the midterm"

The tree-like parser output for noun phrases is suggested by the following structure, which corresponds to the phrase "the longest word."

(NP ((nt word) (sp sing)) (Head.word) (Superl.long))

The "feature" list immediately following the label "NP" indicates lexical and semantic, opposed to syntactic, features that have been built up during a parse. In particular, the nountype (nt) of the head noun is known to be "word," while the singularity (sp) of the phrase has been determined to be "singular." These features are used to enable certain "local" forms of compatibility checks. For instance, adjectives and other modifiers are accompanied in the dictionary by a list of entity types they can modify. When a new word is incorporated into the parse structure, the set of values for each feature attached to it is intersected with the previous set of values for that feature. Whenever the set of values for some feature becomes nil, the parser knows that either (a) the wrong meaning has been used for a word with several dictionary listings; (b) a wrong choice was made at some previous point in the nondeterministic processing of the system grammar; or (c) the input is wrong or anomalous.

The LDC grammar is a hybrid between transition network and phrase-structure grammars.[11,20] Note from Figure 3 that in addition to this grammar the auxiliary files Rout-Word and Compat, each created by Prep, are made use of by the parser. Rout-Word tells which words can begin a syntactic constituent, such as noun phrase, prepositional phrase, relative clause, and so on. Some parts of speech are also included (e.g., to handle the potentially infinite class of ordinals). This information is much like the LL(1) tables of traditional compiler design, and prevents many forms of needless backup during parsing. The Compat file, created by Prep based on information gathered during preprocessing and also on heuristics we have formulated for layered domains, is used to assure that a constituent is "compatible" with the word it is about to be attached to.

Semantic Processing

The job of the *semantics* module is to translate the tree-like parse structures into an internal form that we refer to as "bubble structures."[3] These structures, which can be interpreted directly or can be translated into a formal query for the external retrieval component, possess at least three desirable properties. First, they can be created in a straightforward fashion from the information produced by the previous stages of processing. Second, they capture the idea of the user's input, not merely its syntactic structure. Third, they can be used to direct subsequent processing, which for us means both carrying out actual *retrieval* operations and setting up an appropriate *context* in which to process further inputs.

In determining the proper bubble structure for an input, the domain-structure network, as acquired by Prep, serves as a backbone on which semantic representations are based. Output from the parser is used to complete the representation. For simple sentences, it is sufficient to tag entities in the domain-structure network with the appropriate modifiers from the parse tree. Verbs are treated as adjectives and are also attached to their primary entity, along with their remaining operands. For more complex sentences, that is, those containing relative clauses, a separate semantic representation is built up for each clause. These clause representations are then merged into the final semantic representation. As an example, the bubble structure corresponding to the input "What grade did Mary get in the course John failed?" is suggested in Figure 4. The reader will notice that only the relevant nodes of the full domain-structure network given in Figure 1 have been used.

Evaluation of these semantic representations, or bubble structures, can proceed in two ways. The first method of evaluation, which is discussed in the next section, involves interaction with an external *retrieval* component. In the second method, the bubble structure is used to drive an *internal* interpretation routine. In the latter case semantic processing takes place by producing data representations, or *datareps,* which correspond to the noun phrases of the user's input. For example, the datarep

(section 2 course 3)

would refer to the 2-nd section of the 3-rd course in the domain. Evaluation of the bubble structure given in Figure 4 begins by creating a pointer to the first course in the data file. Ultimately, a reference to each student of each section of that course is made, until either the correct student (John) is found or no more students remain, in which case a pointer to the next section is generated. If all sections for a given course are exhausted, then the next course is examined in a similar manner. When a course is found that satisfies the left-hand side of the bubble structure, the sections of that course are examined for a student named Mary. If Mary is found, her grade is retrieved from the data file and passed down to the "collect" node, and eventually sent to the output generator.

In a pure bubble interpretation, all students of all sections of all courses will be examined to find the student John. Admittedly, this will be an inefficient process. It is important to keep in mind that bubbles are a device to capture the deep meaning of a sentence, not to provide an optimized means of carrying out what has been asked for. Indeed, efficient retrieval almost certainly requires knowledge of the physical structure of the raw-data file, for example which mappings have been "inverted," which we insist on concealing from the natural-language components of LDC. In the interest of efficiency, we provide for an external retrieval component, discussed in the next section.

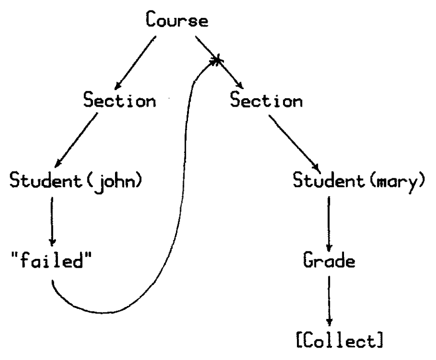


Figure 4—Bubble structure for "What grade did Mary get in the course John failed?"

Output Generation

Finally, an *output generator* converts the top-level datarep produced by semantics into a human-readable form. We are presently using a trivial generator, which for the datarep given above would respond with "CPS 154.2." At a later time we will want to consider methods of generating more informative responses. In fact, we will probably want to incorporate voice response, which is now being provided for NLC by a Votan D5000 device.

THE RETRIEVAL COMPONENT

We have seen that the information acquired by Prep focuses on the *conceptual* rather than the *physical* structure of domain entities. For instance, LDC will know that a course can be "broken down into" sections, but it is not concerned with whether the raw-data file has section names as its "primary key," whether courses are listed with pointers to their sections, and so forth. The actual "binding" of bubble structures and datareps to physical record/table/file structures is handled by a retrieval component, which is regarded as external to LDC proper.

To test the feasibility of this approach, we are having a "standard" retrieval component built (in PASCAL) to interact with the existing LDC code (written in LISP). In addition to routine bookkeeping operations to Get, Put, and Update records, this module will implement certain operations that are standard for layered domains but do not necessarily occur in conventional query situations. These special-purpose operations include *ordinals,* to get the *n*th record with a certain property; *superlatives,* to get the record with the largest, smallest, etc. value in a specified field; and various bubble-type operations such as *averaging* field values, *counting* the number of records in a set, or finding the *percentage* of a set of records having a specified property.

When LDC operates with an external retrieval module, the bubble structure produced by the semantics processor (see Figure 4) is first translated into a formal query. The retrieval component uses this formal query to repeatedly subset the set of all relevant records in the database. At the completion of all the subsetting operations, a value is retrieved from a field or fields of the remaining records. The module also recurses on any embedded query, using the result to evaluate the main query.

For the previously considered input "What grade did Mary get in the course John failed?", the translation into a formal query is suggested by the following.

```
Equal course {Equal student John
              Apply student fail
              Get course}
Equal student Mary
Get grade
```

Evaluation of the imbedded query occurs first, with records in the database that do not have John as an entry in the student field being discarded. Of the remaining records, those that show that John failed, that is made a grade of F, are evaluated for the course name. If two records remain after subsetting, with entries in the course field of EE157 and CPS154, then the main query would become

```
Equal course EE157 CPS154
Equal student Mary
Get grade
```

After subsetting the database using the main query as a guide, only the record(s) with EE157 or CPS154 in the course field, and with Mary in the student field is left. The values of the grade field in the remaining record(s) are sent back to LDC from the retrieval module, whereupon they are passed to the output generator.

DISCUSSION

In designing an NL processor, many decisions must be made regarding where and in what form various kinds of information should be stored. When one is building a single-domain system, many of these decisions can be made arbitrarily or to minimize implementational complexities. When many domains are to be handled, however, unwarranted decisions as to what functions each routine should perform will frustrate efforts to act upon newly acquired information, and may prohibitively limit the flexibility of what can be learned. For this reason, we feel that a learning system has much to contribute to the *theory* of language processing, as well as enhances the technology of natural-language interfaces.

We have seen that the learning mechanisms of LDC allow for *semantic* specifications of new terms. These facilities are important for at least three reasons. First, they allow the user to paraphrase long constructs in a concise way. Second, semantic acquisitions let one ask about certain very complex notions that would be outside the scope of the vocabulary or syntax of the system if they had to be expressed using primitive terms. Third, information is available about new words independent of the syntactic context in which they will occur. For instance, knowing that "large" is an adjective that can modify "section," and that its inflected forms are "larger" and "largest," subsequent processing will be able to accept not only "large section" but also "sections which are large," "larger section than ...," "sections larger than ...," "the largest section," and other forms.

As for technical issues, we have discussed several of the ways in which the design of LDC is *parameterized* to support learning. To oversimplify, learning takes place by converting new information into various forms of *data* to be used by existing processes. Thus, we have designed our semantics processor in a domain-independent fashion, where *meanings* are represented as text to be interpreted, rather than as previously coded procedures. In this sense our semantics module behaves like the parser, which similarly interprets its phrase-structure grammar as a form of data.

ACKNOWLEDGMENTS

The creation of the Co-Occur, Rout-Word, and Compat files appearing in Figure 1 is due to Andrew Reibman, and the implementation of compatibility checking in the parser was done by Nancy Tinkham. These individuals also contributed to our method of producing and interpreting bubbles. We are also indebted to Alan Biermann, Rusty Bobrow, Bill Buttleman, B. Chandrasekaran, Martha Evens, George Heidorn, Gary Hendrix, Bill Ogden, Robert Rodman, and Fred Thompson for valued discussions during the course of our work. Finally, we are grateful to Griff Bilbro, Happy Deas, Linda Fineman, Pamela Fink, and Casey Gilbert for helping to sustain a friendly and productive atmosphere for Natural Language research at Duke.

This report is based on work supported by the National Science Foundation, Grant Number MCS-8116607.

REFERENCES

1. Ballard, B. "A Domain-Class Approach to Transportable Natural Language Processing." *Cognition and Brain Theory,* 5 (1982), 3, pp. 269–287.
2. Ballard, B. and A. Biermann. "Programming in Natural Language: NLC as a Prototype." *Proceedings of the 1979 ACM National Conference,* 1979, pp. 228–237.
3. Biermann, A. "Natural Language Programming." *Proceedings of the NATO Advanced Study Institute on Automatic Program Construction,* Bonas, France, 1981.
4. Biermann, A. and B. Ballard. "Toward Natural Language Computation." *American Journal of Computational Linguistics,* 6 (1980), 2, pp. 71–86.
5. Biermann, A., B. Ballard, and A. Sigmon. "An Experimental Study of Natural Language Programming," *International Journal of Man-Machine Studies,* (1983), to appear.
6. Bronnenberg, W., S. Landsbergen, R. Scha, and W. Schoenmakers. "PHLIQA-1, A Question-Answering System for Data-Base Consultation in Natural English." *Philips Technical Review,* 38 (1978–79), pp. 229–239 and 269–284.
7. Codd, E. F. "Seven Steps to RENDEVOUS with the Casual User." IBM Report J1333, 1974.
8. Geist, R., D. Kraines, and P. Fink. "Natural Language Computation in a Linear Algebra Course." *National Educational Computer Conference,* 1982, pp. 203–208.
9. Haas, N. and G. Hendrix. "An Approach to Acquiring and Applying Knowledge." *First National Conference on Artificial Intelligence,* 1980, pp. 235–239.
10. Harris, L. "The ROBOT system: Natural Language Processing Applied to Database Query." *Proceedings of the 1978 ACM National Conference,* 1978, pp. 165–172.
11. Heidorn, G. "Augmented Phrase-Structure Grammars." IBM Research Report, 1975.
12. Heidorn, G. "Natural Language Dialogue for Managing an On-Line Calendar." IBM Research Report RC7447, 1978.
13. Hendrix, G. "Human Engineering for Applied Natural Language Processing." *Fifth International Conference on Artificial Intelligence,* 1977, pp. 183–191.
14. Hendrix, G. and W. Lewis. "Transportable Natural-Language Interfaces to Databases," *Annual Meeting of the Association for Computational Linguistics,* 1981, pp. 159–165.
15. Mylopoulos, J., A. Borgida, P. Cohen, N. Roussopoulos, J. Tsotsos, and H. Wong. "TORUS—A Natural Language Understanding System for Data Management." *Proceedings of the Fourth International Conference on Artificial Intelligence,* 1975, pp. 414–421.
16. Plath, W. "REQUEST: A Natural Language Question-Answering System." *IBM Journal of Research and Development,* 20 (1976), 4, pp. 326–335.
17. Thompson, F. and B. Thompson. "Practical Natural Language Processing: The REL System as Prototype." in M. Rubinoff and M. Yovits (eds.), *Advances in Computers,* Vol. 3. New York: Academic Press, 1975.
18. Thompson, F. and B. Thompson. "Shifting to a Higher Gear in a Natural Language System." *AFIPS, Proceedings of the National Computer Conference* (Vol. 50), 1981, pp. 657–662.
19. Waltz, D. "An English-Language Question Answering System for a Large Relational Database." *Communications of the ACM,* 21 (1978), 7, pp. 526–539.
20. Woods, W. "Transition Network Grammars for Natural Language Analysis," *Communications of the ACM,* 13 (1970), pp. 591–606.
21. Woods, W., R. Kaplan, and B. Nash-Webber. *The Lunar Sciences Natural Language Information System: Final Report,* Bolt, Beranek and Newman Report 2378, 1972.

Implementation of an Ada* run-time environment

by HERMAN FISCHER
Litton Data Systems
Van Nuys, California
and
EDGAR H. SIBLEY
Alpha Omega Group, Inc.
Silver Spring, Maryland

ABSTRACT

The Ada Programming Support Environment (APSE) has been introduced[1] as a set of tools to support program development systems. This paper introduces the idea that the concepts and facilities of APSEs are valuable not only to host systems (those used to develop software), but also to certain Ada run-time environments (ARTEs) (those in which applications execute) and examines the implementation of large database transaction oriented systems in such an environment. Two examples of actual systems are used to show the benefits gained by using the Ada environment. A cost/benefit analysis for such a transition is also outlined.

*Ada is a registered trademark of the Department of Defense.

INTRODUCTION

The benefits of using an Ada run-time environment (ARTE) based on the Ada Programming Support Environment (APSE) model are basically the same as the APSE's benefits for the programming host: transportability and interoperability of system components in a hardware-independent manner that was not previously realizable. These terms, though in common use, are not always used in the same way and must first be defined. The KITIA[2] have agreed that "interoperability" is the ability of support environments to exchange database objects and their relationships in forms usable by tools and user programs without conversion, and that "transportability" is the ability to install a function in a different environment, without reprogramming, to perform with the same functionality.

The ARTE/APSE Concepts

It is our belief that the STONEMAN model, though defined for host/target environments, will be applied to computing environments not intended for program development. In STONEMAN, a kernel APSE (KAPSE) is surrounded by a Minimal APSE (MAPSE) toolset that, when extended with a comprehensive set of tools, becomes a full APSE. This is illustrated in Figure 1, with the KAPSE at the center of a set of rings (embodying the hardware, operating system, standardized interprogram communications mechanisms, and its "database"), and with the surrounding wedges representing the minimal toolset. Surrounding the outside of the wedges are applications-specific tools and programs; these can also occupy "open" (MAPSE) tool spaces in the wedges ring.

It is proposed that the layered-rings model applies to transaction-oriented database management system (DBMS) ARTE systems, as portrayed in Figure 2. Although more detail is shown in the kernel of the ARTE (KARTE), the case will be made that the features of the KARTE are applicable to the APSE also. In fact, an ARTE can use many of the same facilities, or share the same processor with an APSE (as happens today in logistics systems that run applications software in the processor used to support software-development organizations).

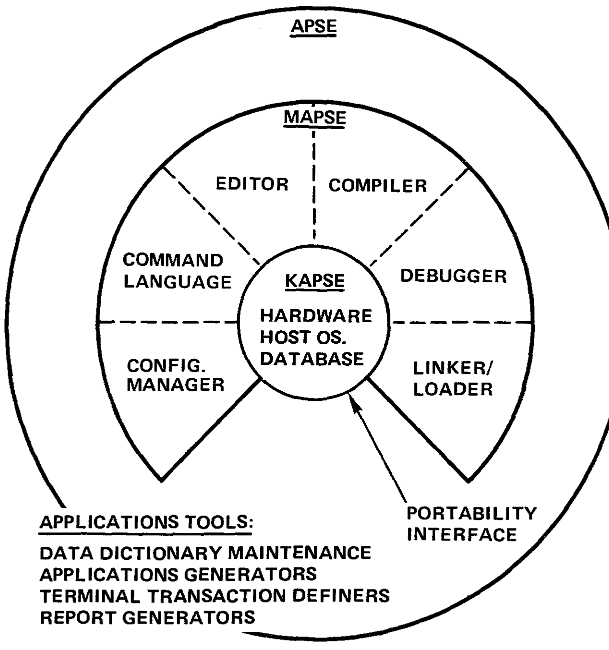


Figure 1—STONEMAN model of APSE

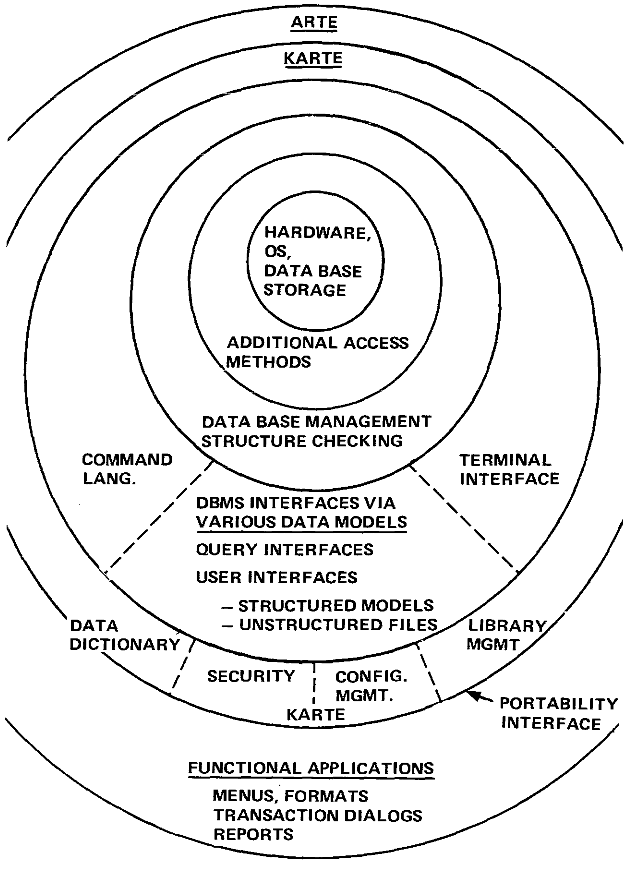


Figure 2—Application of model to transaction-oriented DBMS ARTE

We shall first examine the need for this form of run-time environment.

Problems of Existing Large-Scale Information Systems

There are six categories of problems that exist in current systems; we shall first show how these are solved in a current programming and run-time environment (there are also many problems with existing systems that are only remediable with improved software methodologies, better maintenance, more money, and the like). These are:

1. *Hardware*—Hardware, as used in large systems is often obsolete before the applications software is placed in operation; in fact this may occur early in the application's life cycle. Hardware is also often not suitable for new, heretofore unthought-of applications of existing software, such as occur when rapid-deployment concepts force portability of previously stationary systems.
2. *DBMS*—DBMSs are large, complex beasts. The successful ones marketed to Fortune 500 companies seem to have upwards of 2,000,000 source lines, including the various optional (but essential) tools that surround the DBMS. These tools are constantly evolving, and upgrading from one major version to the next is a major undertaking. Furthermore, as the DBMS systems evolve, new generations of systems appear. In the absence of KAPSE-like layers separating applications from underlying implementations, portability is not possible. Existing applications are enormously expensive to upgrade.
3. *Tools*—DBMSs contain, in their environments, a large quantity of very diverse tools; these vary from initialization, backup, and restoration tools, to online dictionaries and schema-maintenance tools, reporting and configuration-management tools, applications generators, Query tools (structured and English-like), and so on. It is not reasonable to switch tools every time new hardware is obtained, or when switching to an improved underlying DBMS.
4. *Applications Generators*—Such applications generators as the Cullinane Corp ADS-OnLine,[3] the IBM IMS-ADF,[4] and the Air Force On Line Data System (AFOLDS) DUEL[5] are a vital part of today's environment. These products allow transactions to be coded in very high level applications languages, or to be expressed as a set of rules or decisions. The Cullinane product is said by commercial users to provide a sevenfold to tenfold productivity improvement over "old" (manually-coded) programming techniques. The USAF product's users estimate a 60% improvement in productivity. Applications generators are expected to become a vital part of the future of DBMS-based transaction processing systems, even though they are not, strictly speaking, DBMS tools. Thus, their future use will need some attention to the portability of the applications-generator toolset independently of the underlying DBMS or hardware.
5. *Transaction-Based Terminal Handlers*—The ability to create user-friendly and easily maintainable transaction-processing software depends, to a large extent, on user terminal handling processors that are intimately integrated into the underlying system. At the same time, they must be sufficiently separable to permit migration among significantly differing terminals, networking philosophies, communications facilities, and report generators that supply the user needs. Standish discusses the needs for a bounded set of user interfaces for accessing tools and the possible need for standardization in this area.[6] User interfaces have been categorized as "normal conversational" (prompted command lines), "form filling" (CRT with prompts and fill-in fields), "tree of menus" (hierarchy of levels of choices), and "graphical" (windows and iconics). All these will be needed at future user interfaces.
6. *Configuration Management*—Configuration-management features are an integral part of today's complex large-scale transaction-based systems. However, they are often treated lightly and incompletely; the result is that logistics-type systems have never had great success in supporting variants and versions of schemas, programs, and transactions. Nearly all major DBMS and their dictionaries today support identification of versions or variants, and yet almost none of these support the coexistence or automatic maintenance of different versions at different sites, exchanging distributed data, converting formats, ensuring proper library control and testing, and recovering crash and archival data of differing versions.

SOME EXISTING SYSTEMS

The U.S. Air Force relies on two very different large-scale DBMS-oriented logistics systems for aircraft maintenance tracking.

- The Maintenance Management Information Control System (MMICS), presently implemented on the Burroughs Medium System computer family (the Phase II system) handles fighter aircraft (F-15, F-16, and A-10). This system tracks engine parts and operating conditions, and performs calculations to predict the need for preventative maintenance.
- The Automated Maintenance System (AMS), based on the IBM 370 family architecture, handles airlift transporters (C-5, C-141). It has also been considered for handling logistics for the B-1, the Space Shuttle (if the USAF takes responsibility for it), and the MX system.

The MMICS system is huge, encompassing far more than fighter engine maintenance tracking; MMICS provides much of the data required to manage maintenance equipment and personnel resources, worldwide, for aircraft, missile, and communications-electronics-meteorological environments. MMICS includes over $\frac{1}{2}$ million source lines of coding.

The AMS system is also huge, including a large number of

TABLE I—Systems Environments and Problem Areas

	Example Transaction-Based Logistics Systems		Current Environments			Transportable Ada-Based Environment
	MMICS (Phase II)	AMS	Phase-II Compatible	IBM Compatible		
				IMS	Cullinane	
Purpose/Application	Engine maintenance & Logistics F15, F16, A10 Aircraft	C5, C141 Aircraft				
Hardware	Burroughs B3500, B4700	IBM 360 Compat.	Burroughs B3500, B4700	IBM 370 compat.	IBM 370 compat.	KAPSE compatible
DBMS	Internal Cobol (none)	IMS	AFOLDS	IMS	IDMS	Adaplex (CCA Corp.)
Tools	Special Cobol	IMS utilities	Backup restore Data Dictionary Misc.	ADF utilities	IDD	–
Applications Generation	None	None	DUEL	ADF	Adds-On Line	–
Terminal Formating	None	IMS macros	FRAMES	ADF	On Line Mapping	–

components and users. AMS is more recent than Phase II; AMS is architecturally an online system, based on the 327x terminal family and IBM's IMS database system.

MMICS is an "updated system;" its COBOL programs reflect batch processing punched-card transactions and a "home-grown" database structure (embedded in the transaction programs). MMICS has since evolved to be online, but only by overlaying on the Burroughs Master Control Program (MCP) a USAF "home-grown" transaction-analyzer program (itself over 110,000 source lines) to simulate punched-card inputs and line-printer output on CRTs.

MMICS operates in a base support computer environment, where the complete system is known as Phase II. Not all Phase II programs are COBOL-based with embedded database handling: the Civil Engineering, Accounting and Finance, Medical, Operations, and Transportations applications are all implemented (on the same B3500 hardware) using AFOLDS, which includes data description capabilities, an applications generator, a transaction-oriented terminal handler (and forms builder), and an English-like query language.

Neither Phase II nor AMS is transportable; Phase II is not because its MMICS has a large volume of embedded assembler coding within Cobol programs to handle database access. Phase II's AFOLDS-based functions are also machine dependent. AMS is not transportable because of the use of IMS and 327X terminal formatting facilities.

Table I examines these systems with respect to the problem areas enumerated previously.

Hardware and Transportability

Phase II is tied to Burroughs Medium-Sized Systems. Most bases have "ancient" B3500 computers installed. These transistor and discrete component curiosities cannot be replaced, under Government Accounting Office rules, except by competitive bidding. (A few bases have slightly newer B4700s.) Although Burroughs makes modern equivalents, such as the 900 series B3900, the USAF has not been able to convert to them (due to the obvious lack of competition) and is thus stuck with the B3500s.

Phase IV, the upgrade-in-process for Phase II, was directed by public law to be a competitive procurement (the authors do not mean to imply that it is bad, only that STONEMAN concepts are needed to make such future actions reasonable). Two companies are in a "compute-off" conversion, Burroughs and Univac; both are re-implementing the present system on new (different) hardware families, using new operating systems and terminal handlers, and using manufacturer-supplied DBMSs.

Even Phase IV hardware will become obsolete in the near term (owing to the pressures of an advancing semiconductor industry), and at that time the DBMS, operating system, and terminal handling systems will remain untransportable. The Phase IV system, once deployed, will be no more readily transportable (in competitive reprocurement) than Phase II.

AMS, being based on IBM architecture, appears to have a longer life: there is a large industry of instruction-set-compatible processor builders. AMS users can thus develop new software and not worry that near-term hardware upgrades will remove the IMS and 327X terminal architectural footing.

Independent Systems and Interoperability

There will always be organizational, spatial, and temporal reasons for independent procurements of systems with similar requirements. For example, the aircraft maintenance software for fighter aircraft and transport aircraft has been handled by separate organizations, and it is therefore not surprising that MMICS and AMS are two different systems. MMICS is over ten years old. If it were not for its lack of transportability, some of the MMICS sites would have been upgraded to more recently procured hardware. However, unless the new system remains interoperable with the old one (in addition to being able to reuse its software), there is no way to phase-in new equipment, and there would thus be no way to reduce acquisitions of side-by-side systems such as MMICS and AMS which could otherwise share resources.

ENVIRONMENT CONCEPTS APPLIED TO THE EXAMPLES

Both the Phase II and AMS systems will, at some time in the future, need to be transferred to new-technology support systems. There is only one cost-effective way that this can be accomplished within the constraints of implementing transportability and interoperability. Software costing studies show that a robust modern support environment, providing a toolset which minimizes the complexity of applications programs, minimizes costs.[7] Clearly there are two prerequisites to such a transition: a robust DBMS and a complete set of tools to support its use (e.g., applications generators).

Given that these prerequisites were met on an Ada-supportive system, one would still be faced with the problem that the resultant new systems would be nontransportable (except among families of the Ada-supportive system chosen). For example, if one were to base a reimplementation on the ALS architecture (the U.S. Army Ada Language System effort), the system would be tied to DEC VAX architecture and its current operating systems. Furthermore, the DEC architecture, as it is being used by the Ada products, is not supportive of transaction-based terminal networks (e.g., multidropped externally clustered buffered approaches).

An attempt to build a run-time system for the transported systems is likely to meet with two barriers, the cost and the lack of guarantees of support in future environments.

The only solution is to base the transported system on the use of the STONEMAN concept, where the transaction-processing software itself is an outer layer of programs around a "ring" of DBMS and transaction-supportive tools.

COST/BENEFIT CONSIDERATIONS

The Phase II to Phase IV conversion, now in progress, provides interesting cost data. Each vendor is charging the taxpayer (in round numbers) $50 million for the first increment of software conversions. In addition, the USAF has approximately 500 staff members supporting the three-year effort.

Given that 1500 man-years cost approximately another $50 million, the initial conversion cost is of the order of $150 million—owing to the lack of software transportability. The initial nine applications converted (of several hundred) are said to be the most difficult, and may thus represent as much as half of the total effort.

Moreover, Phase II is only one system, within one service, and the selected hardware, however good it is at the time of delivery, will be hopelessly obsolete by the end of the decade and, under federal rules, unreplaceable except by a new multi-vendor competition.

In an ARTE, only the kernel would need re-interfacing to transport a system; that would be some orders of magnitude less expensive to the taxpayer.

RELATIONSHIP BETWEEN APSE AND RTE

Where would one find an ARTE? Certainly the most obvious possibility is to utilize the structure of an APSE and replace some of the tools in the MAPSE with transaction and database oriented functions, and utilize the same core KAPSE for both APSE and ARTE.

This idea's merit becomes clearer when one looks at the systems given as examples above, and realizes that, in both cases, the identical operating system (the major part of a KAPSE) is used by both the programming-support centers and the operations centers; in fact, operations often shares the same processor with development on a time sharing basis.

It is then worth examining whether a KAPSE can also be the kernel for an RTE.

Differences in KAPSE Databases

The STONEMAN document is a generic statement of the goals of PSEs. Therefore, it is not unexpected that "instances" of environments supposedly designed to meet these goals differ substantially in matters important to the use of a KAPSE for an ARTE kernel. Indeed, the definition is so loose that it is possible to produce conflicting (and definitely nontransportable) software based on different KAPSE implementations.[8]

Key areas are in the database support mechanism. The SofTech Ada Language System implements a hierarchical "database" structure based on the concept of trees of nodes, attributes, stream storage (unformatted), etc.[9] This database structure is well suited for "bulk" storage (of program text, compiled code, and unstructured small files). The Intermetrics Ada Integrated Environment[10] implements a "semirelational" sort of database that uses a mechanism similar to directory trees to locate each stored element (attribute, unstructured file, or indexed file). Both of these environments basically place the responsibility for the structuring of data with the tools and programs that lie outside the KAPSE (e.g., compilers, program library managers, etc.). Two European efforts (the UK[11] and the EEC[12]) place in the KAPSE the structuring mechanism for program support access to code trees, program-library configuration data, and so on.

Other key areas include "options" such as configuration management. The SofTech and Intermetrics effort both place this responsibility as an embedded function within using tools, rather than as a kernelized system service. (An effort to implement configuration management in a transportable and interoperable manner is presently funded to CSC; however, this is being provided as a MAPSE tool, rather than being integrated into the run-time environment.)

A Structured Database Project for the KAPSE

There is one known attempt to place a "structured" database onto a KAPSE environment. The continuing success of this effort could be used to base an argument that KAPSEs, in general, are capable of supporting the form of database required by transaction-based logistics systems, such as the two discussed above.

The Computer Corporation of America's Adaplex effort[13] is developing an Ada-compatible DBMS. Their approach is based on an entity and function database structure, with typed data. Adaplex uses a database model said to furnish more capabilities than either the hierarchical, network, or relational database models. Although Adaplex is being implemented to run on a DEC VAX computer, using the SofTech KAPSE and MAPSE toolset, the authors of Adaplex intend to use the VAX VMS operating system calls to access the disk directly (instead of utilizing the SofTech KAPSE for disk access).

Although Adaplex is coded in Ada, it will be non-transportable because of its VMS dependencies. The SofTech KAPSE could probably be redefined to include the appropriate disk-access services. Yet even given that the KAPSE contains the necessary services, the SofTech KAPSE is non-transportable because it is heavily dependent on VMS services and non-Ada internal coding.[14]

The KIT[15] and KITIA[16] have efforts to create standardized KAPSE interfaces; fruition of interface standardization would allow the Adaplex DBMS to be transported to other KAPSEs.

Given that (a) the Adaplex can eliminate VMS dependencies by a change to SofTech's KAPSE services and (b) the KAPSE interface-standardization efforts meet with success, one would postulate that Adaplex could become an instance of a transportable DBMS usable for constructing an ARTE. This example would place the DBMS in the tool layer surrounding the kernel, not in the kernel itself.

Using a Non-Ada DBMS in an APSE

The enormous cost of implementing a DBMS from scratch, along with the necessary "optional" tools (applications generators, transaction terminal handlers, etc) has led to the study of incorporating existing, non-Ada DBMS implementations with Ada environment tools and programs. A proposed solution[17] allows structured queries and updates from Ada packages to map into non-APSE DBMS primitives, by translating the operations and binding the data. Thus, a set of package interfaces "crosses" the APSE domain into "foreign" DBMS facilities, without going through a standard KAPSE interface. Existing DBMS tools, application generators, terminal handlers, and English-like or well-designed query tools of the original environment are not available to the APSE user, except through the underlying operating system. This shortcut solution thus does little to provide for transportability and interoperability at the DBMS application level.

USING AN ARTE DBMS IN THE APSE

Conventional DBMSs do not make a good job of dealing with bulk data in the form of the so-called unformatted file or in the form of libraries of nonhomogeneous data, such as a set of programs that have been partially or fully link-edited as a system ready to be executed. Such data occurs in streams of bits representing machine structures: words, or bytes, or paragraphs, or syllables, or blocks, and so on. The stream or its parts may be directly or randomly accessible, or it may only be serially accessible. Such data are sometimes called unstructured data, and the database said to have no knowledge of the internal form of the data. This may be true in some cases, but it does not capture the essential difference. For example, in Ada, a file is said to be "associated with an unbounded sequence of elements, all of the same type." It can be argued that the system is required to know the element type, to ensure that all users access it using the same element type.

A suitable treatment of bulk data is essential in the KAPSE, because the entities that are controlled through a programming support environment (PSE) are primarily associated in storage as bulk text (e.g., Ada source and compiled objects). The normal way to deal with bulk data in the past has depended on the usage of that data. If the data were an entire system, a program, or a part of a program, and so on, it was placed in a "library" that could access the program-unit by its name. The structure of the program-unit was generally simple or nonexistent. If the data were to be accessed by a procedure, they were stored as a relatively conventional set of records in a file or in some similar fashion (indeed, they could be stored as a stream of characters or even as a stream of bits, but the procedures and the supplied access methods were the only way that the data structure was known). One of the special Ada data structures is, of course, the Diana tree. This has a structure that has been standardized, and the fact that the Diana tree has needed to be made a standard is an interesting example of the need for standards within the KAPSE or with the MAPSE levels—in order to allow inter-tool action at that level.

Clearly, if a DBMS is to be provided for use by the PSE, it would be very convenient and useful if the same DBMS were also suitable for applications use. This would mean that there was a DBMS in the KAPSE to allow for the addition of a formatted data concept to the PSE; as a result, the DBMS could be used for other functions (as discussed later) and if the same DBMS were to support such actions as an ad hoc query, then the KARTE would potentially have the same interface DBMS.

Classes of Data Supported

As already discussed, there are at least three classes of data that are important in the Ada environment. These are:

1. *Unformatted data.*—This is a broad class of data that may have structure, but which have no structure that may be known to any program or procedure except through special communication from a programmer. This class of data could be a report in character form (possibly internally indexed, or part of a word processing system with a retrieval mechanism), or it could be a traditional file (with a well-defined file structure that needs a special access method—such as ISAM—to act as

an indexing device for rapid retrieval), or a "bucket of bits" that might be the results of a transmission or a program. All types within the class of unformatted data have one characteristic in common—they consist of a stream of bits that may have structure, but this structure is unknown outside the suite of procedures that access the data.

2. *Standard formatted data.*—Such data have a predefined format that has been previously defined by a community of users who are on their honor to see that all tool-using devices are used consistently. Typical of these data are the groups of procedures that work on a common data structure. In a run-time environment these may be a suite of personnel programs that provide accounting, payroll, and personnel support services; in Ada the compiler and editor interactions (via Diana trees) are examples of standard formatted data systems. The difficulty with such systems is that they rely on the good will of the users (or the hard heads of auditors) and often are violated, either deliberately or in error.
3. *Fully formatted data.*—These are seen in database-managed systems and in some structured PSEs. The essential characteristic of such data is that the environment is aware of the structure of data as a whole and of any part of them. For example, the PSE being designed in the UK has a built-in structure which assures that the program parts are properly controlled—thus the subprograms of a main program are associated with it; moreover, the particular version of the subprogram that is valid with it. Thus the "structure" of these data is an exact match of the configuration that must be managed. The run-time environment, then, will be able to retrieve the required version of the main program and with it all relevant and correctly versioned copies of its subprograms.

APSE AND ARTE INTERACTION

The ARTE and the APSE are not really easy to differentiate, yet an attempt to do so for the programming environment has led to the definition of an entity termed a KAPSE. And it appears that in the time since the STONEMAN document was approved the Ada community has forgotten that the prime reason for a programming environment is to provide programs that can be run—presumably in a run-time environment. The idea of a host-to-target environment, of course, has led to some of this apparent neglect, but most target machines have a need for some environment, and it seems reasonable to assume that future target-machine architecture will benefit from the definition of a standard run-time environment. Naturally, any other run-time environment that interacts with other machines/computing devices will benefit even more if a standard KARTE exists.

Much then has been said of the KAPSE, but little of the KARTE. If, as appears very likely, the Ada langugage is used to implement logistic and other large-scale nonoperational (large-scale administrative) systems, then some of the requirements of a KARTE that were only marginally necessary in the KAPSE will be more essential. These features include the need for security features; a way to store data structures and define the meaning of the data entities (an information-resource dictionary); a means for storing and retrieving data based on these definitions (a generalized DBMS with good user interfaces for query, table generation, and reporting); methods for recording program structures, their relationships to data and users, and so on (a good software-configuration-management system); and interfaces to the documentation, which can be a part of the combined configuration and information resource management system.

There are at least three types of security that should be investigated: user checking, procedure validation and initiation, and data sensitive checking.

The DBMS-like features should include the capability of interfacing to a dictionary. Modern dictionaries have many different features, but they are all generally able to capture compiler data to document data and program usage. Some are even able to hold information on the users, some security needs, and some configuration-management controls. The controls between a DBMS and a dictionary, and even the configuration manager, are implemented through an active interface between the dictionary and its users (automated or human). The value of the dictionary is only fully realized if it can operate in this controlling role.

CONCLUSIONS AND A PROPOSED ARCHITECTURE

The purpose of this paper was twofold:

1. To show that there was a need to consider the use of the KAPSE as an ARTE.
2. To suggest some of the problems in making the transition from one to the other.

It does seem, however, that there is a possibility of using architectures that have been developed for operating systems, DBMSs, and information resource dictionary/configuration management systems in the past. One possible high-level architecture was given in Figure 2.

This shows that it would be possible to use the general architecture of the STONEMAN KAPSE for a combined programming and run-time support environment, but some additional controls will need to be added to the system, and the configuration management system would need to have control of any access to the libraries.

However, the use of an Ada environment that had realized these features would make future software more easily transportable and allow real "software reusability," thereby reducing the rising costs of software while allowing major systems of the future to be implemented in spite of the expected "gap" in available programmers and systems implementors.

ACKNOWLEDGMENT

The authors are grateful to Mr. Leo Meany of the U.S. Air Force Data Systems Design Center, who provided information on AFOLDS.

REFERENCES

1. Department of Defense. *Requirements for Ada Programming Support Environments "STONEMAN."* Department of Defense, February 1980.
2. "Definitions." *Kernel Ada Programming Support Environment (KAPSE) Interface Team: Public Report Volume 1.* San Diego, Calif.: Naval Ocean Systems Center, Technical Document 509, April 1982, Appendix C.
3. *Application Development System/OnLine Reference Guide.* Cullinane Database Systems, Inc., Order Number TDAO-330-10, January 1982.
4. *IMS/Applications Development Facility Program Description/Operations Manual.* IBM Corporation, order number SH20-1931.
5. *AFOLDS, Air Force OnLine Systems User's Guide.* U.S. Air Force Data Systems Design Center, Gunter AFS, Alabama, 1975.
6. Standish, T. "Extensibility." *Kernel Ada Programming Support Environment (KAPSE) Interface Team: Public Report Volume 2.* San Diego, Calif.: Naval Ocean Systems Center, 1982.
7. Boehm, B. *Software Engineering Economics.* New York: Prentice-Hall, 1981.
8. Fischer, H. "Time Line Analysis of KAPSE Interfaces During a Compilation." *Kernel Ada Programming Support Environment (KAPSE) Interface Team: Public Report Volume 2.* San Diego, Calif.: Naval Ocean Systems Center, TD552, 1982, pp. 3Q-1 ff.
9. "The Environment Database." *Ada Language System Specification.* Waltham: SofTech, CR-CP-0059-A00, June 1981, Appendix 50.
10. *Computer Program Development Specification for Ada Integrated Environment: KAPSE/Database.* Cambridge: Intermetrics, Inc., Draft IR-678, June 1981.
11. *Final Technical Report. KAPSE Database.* London: United Kingdom Department of Industry.
12. Gallo, F. Presentation to EFDPMA Ada Conference. Copenhagen: Olivetti Corp., September 1982.
13. Smith, J. *Reference Manual for AdaPLEX.* Cambridge, Mass.: Computer Corporation of America, 1981.
14. "Rehosting." *Ada Language System KAPSE B5 Specification.* Waltham: SofTech, CR-CP-0059-C81, August 1981, pp. 3–63.
15. Kapse Interface Team. "Ada Programming Support Environment Requirements for Interoperability and Transportability and Design Criteria for Standard Interface Specifications." *Kernel Ada Programming Support Environment (KAPSE) Interface Team: Public Report Volume 2.* San Diego, Calif.: Naval Ocean Systems Center, TD552, 1982, pp. 3G-2 ff.
16. Sibley, E. "Towards a KAPSE Interface Standard." *Kernel Ada Programming Support Environment (KAPSE) Interface Team: Public Report Volume 2.* San Diego, Calif.: Naval Ocean Systems Center, TD552, 1982, pp. 3K-1 ff.
17. Bever, M., M. Dausmann, S. Drossopoulou, W. Kirchgassner, P.L. Lockemann, G. Persch, and G. Winterstein. "The Integration of Existing Database Systems in an Ada Environment." *Proceedings of the AdaTEC Conference on Ada,* New York: ACM, 1982.

Future Ada* environments

by SABINA H. SAIB
General Research Corporation
Santa Barbara, California

ABSTRACT

The current Ada environments are oriented toward traditional code production tools such as editors, compilers, loaders, and program library managers. Future Ada environments will add to the initial capabilities to provide support from the initiation of requirements to the enhancement of existing operational software. In addition to software development facilities, future Ada environments will support management activities. The future will also see applications of current tools and techniques across the entire life cycle.

*Ada is a trademark of the Department of Defense.

INTRODUCTION

Ideally all phases of the software development cycle from requirements to maintenance or enhancement should be supported by a software environment. Many current software tools contain capabilities for supporting one phase of the life cycle. These tools ignore what has gone before in the life cycle or what will happen in the next phase. Often different languages that are oriented towards a portion of the life cycle are used, so that the tool is totally incompatible with all other tools in use. In the use of such tools, much effort is expended in transferring incompatible data from one tool to another. This effort could be avoided by implementing an integrated support environment. During all phases of software development, management needs to be able to see what the status of a project is. Today this is often done without the use of tools that can look at the actual state of the software. As a result, the management view is often incorrect and management is unable to address problems in a timely manner.

LIFE CYCLE CONSIDERATIONS

The traditional approach to identifying the capabilities needed in a software development environment has been to examine each phase in the software development life cycle as a separate activity. Occasionally its relationship to immediately preceding and following phases is also considered. Recently emphasis has been placed on the ability to trace requirements through specifications and design to the implemented code elements and acceptance tests. Another approach to developing an environment for life cycle support is to examine the needs of the maintenance or enhancement phase. The impact of a change in the requirements must be traceable out to the affected software components, and the proposed changes to the software must be traceable back to the unaffected requirements.

GOALS FOR AN ENVIRONMENT

Any future Ada environment must include a wide variety of capabilities in order to support the development of software during all phases of the life cycle. It must be easy to use, and assist the user not only in such detailed activities as interactive debugging, but also in organizing and directing the effort. The environment must be extensible to allow the addition of capabilities as new tools and techniques emerge, and it must be standardized across machine architectures, operating systems, and file access methods. The capabilities for an environment can be implemented as a set of cooperating development and management tools. These tools can be clustered into three major groups: multipurpose, software production, and management.

Multipurpose tools find use in numerous phases of the software life cycle. These tools for examining and updating text files and the generation of reports must be designed with general purpose capabilities to reflect their universal usage. The most important multipurpose tool that can be developed for an environment is the database manager. The Ada environment requires a single database for all environment activities. Such a database will automatically provide a history of a project and avoid duplication of database functions among tools.

Software production tools that work together can form a comprehensive package of support for a user. Some specific tools that should be developed for the future Ada environments are requirements processor, specification processor, design analyzer, coding assistant, standards checker, compiler, static analyzer, linking loader, configuration manager, test assister, and verifier.

Management tools fill needs that differ from those of the software production staff. Activities that should be supported by tools are planning, staffing, controlling, directing, organizing, and status reporting.

From a designer's viewpoint, the goals imply that the Ada environment must be both extensible and modifiable. Experience has shown that nearly all truly useful systems grow and change over time as new needs are developed and old ones become obsolete. Extensibility and modifiability are enhanced by the use of a single, uniform, functionally oriented command language. Tools will be contributed to the Ada environment from numerous sources. The sheer number of tools that should be provided makes it impossible for one group to be the sole suppliers to the environment. Furthermore, the particular tools within the environment will be changing. A tool should rely less on the physical format of information than on its logical structure. A uniform database system and database manager provide such a capability. The virtual memory manager in the Ada Integrated Environment appears to offer this capability. The design of a new tool can ignore which other tools in the environment create the information it needs; it only needs to know that the data will be created and will be accessible though a standardized format that stresses the logical rather than the physical characteristics of the data.

SOFTWARE DEVELOPMENT DATABASE

For each software project, a common information storage and retrieval system should provide a repository to consolidate all relevant project data. The database also serves to unify the tools in the environment by providing common access to project data. The project database must, therefore, be common to all tools in the environment. Each tool should use the database access facilities to retrieve the data it requires and to store the information it derives. The use of such a database is shown in Figure 1.

The advantages of a common database are many. Information required by more than one tool can be computed and stored once, avoiding duplicate data files and extra processing. Also, information can be conveniently passed from one tool to another, communicating through the database. Information managed within a database is more reliable than data scattered in separated files. Information is less likely to become inconsistent, because the database can impose a number of consistency constraints. For example, analysis data may be inconsistent if a program has been modified since the analysis was done. Such analyses can be marked as obsolete when a program has been modified, so that reanalysis can be scheduled at a convenient point in the development.

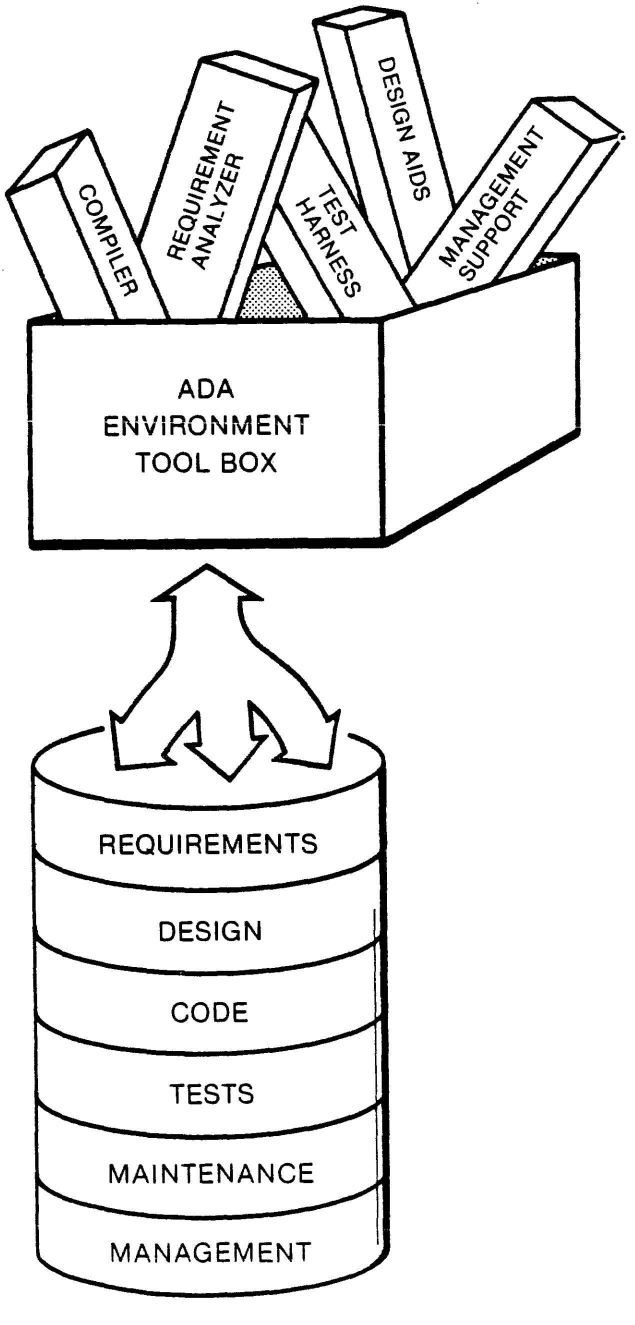


Figure 1—Future Ada environments

REQUIREMENTS AND SPECIFICATION TOOLS

The requirements state what a computer system should do from the user's viewpoint. The environment should aid the user or systems analyst who must enter the requirements in machine readable form and must aid the analyst who must convert the requirements into a system design.

There are five properties that a well-written requirements document should have; it should be

Complete:	say everything the implementor needs to know.
Consistent:	not contradict itself.
Testable:	implementor can objectively determine when the job has been done correctly.
Unambiguous:	implementor can interpret requirements in only one way.
Concise:	not ramble on.

To a certain extent the environment can help an analyst judge a set of requirements with respect to these criteria. In addition, it is possible to construct a requirements definition system that prompts the user for information so as to encourage requirements with these desirable properties and also to enhance the environment's ability to detect flaws.

Requirements are complete when all system inputs and outputs are fully characterized, system level error policy is stated, all documentation and deliverables are specified, and the functional relationships between inputs and outputs are stated. For inputs and outputs this requires specifying the data type, the value range, a prose description, a mnemonic name for reference, the source for the input and destination for the output, and its format. The environment can prompt the user for such information about each proposed system data item. This will help ensure that the requirements are complete. Furthermore, since the environment obtains the requirements through an interactive dialogue, the information will necessarily be in machine-readable and machine-analyzable form.

For system errors the user should decide on the error handling capabilities of the system. Error processing is too important to be left in the hands of systems analysts. The analyst's job is to determine the feasibility and cost of the desired error processing capabilities. For each error condition the requirements tool should prompt the user for a description of the nature of the error, what the user would like to see displayed to indicate detection, such as a lit-up panel or a sounding alarm, and what recovery action to take, such as turning off a sensor.

The functional relationships between input and output should be specified in a moderately nonprocedural way, since the user who will supply them cannot be assumed to be a systems analyst. SADT,[1] a manual system from Softech for requirements definition, is a nonprocedural graphical approach to system decomposition. Graphical support for requirements definition seems highly advantageous because a clear drawing can offer a better perspective on logical relationships than simple words can. An automated version of SADT that is merged with a prompting system as just described would be a valuable requirements tool.

In addition, the requirements tool should request a schedule of activities, deliverables, tolerances whenever approximate answers are possible, timing constraints, physical constraints, budgetary information, equipment to be used, applicable standards, testing practices, and acceptance procedures. By having the tool explicitly request this information, which will be primarily prose, hence not very analyzable, the system database will be able to index the requirements automatically for each of these major categories. Later, if the user wishes to request the set of applicable standards for the project, the database manager will be aware of which part of the requirements dealt with this subject. With a less structured format for entering requirements, such queries would be more difficult, if not impossible, for the database manager to successfully respond to.

If the requirements are entered in the fashion just described, it may be possible to perform limited tests for consistency. RSL,[2] the TRW requirements specification language, has such a capability. With the limited information about the system available in a requirements document, it is still able to check, for example, that all input items are used in the computation of some output value, and that no output item is also treated as an input item at a different point in the system.

Another advantage of using an automated tool to enter requirements is that it simplifies tracing requirements into the design and code. The Requirements Tracing Tool[3] developed by Logicon tries to achieve this by forcing requirements to be written in a format in which each identifiable requirement is tagged with an indexing number that is explicitly written into the design and code implementing the requirement. With the requirements tool described here, the tracing of requirements to design is automatic because the design will be generated automatically from the requirements. The system can report the relationships between design elements and those aspects of the requirements from which they were generated.

SOFTWARE DESIGN

A program design should not only be useful in the stage of the life cycle between the requirements phase and the implementation phase of a project, it should also aid the designer in expressing the design, aid a reviewer in checking the design, aid an implementer in developing the design into a product, aid a tester in validating the resulting software, and aid a maintainer in changing the design.

There are at least two levels of detail that a design language should support: programming in the large and programming in the small. In programming in the large, the user should be prevented from detailed design. It should be possible to use the requirements database to develop automatically a skeletal system design that contains the highest level modules written in pseudocode. Module inputs and outputs, a pseudocode description of purpose, the attributes of the inputs and outputs, and the interfaces between modules can all be automatically generated from the information gathered through the dialogue described earlier. The analyst can restructure this skeletal design and enhance it with the further details that are inappropriate for the requirements.

In the design stage, a user should be able to input fragments of a design and receive information on what has been entered, consistency checks on what has been entered, and if desired a measure of the completeness and complexity of the design description. Analysis of a design should proceed in stages such that the user is not inundated with information on the entire design while the design is incomplete. Analysis reports should be interactive, with the designer able to quickly alter the design, reanalyze that part of the design, and view the results on an interactive basis. The design analyzer should produce consistency reports at various levels of detailed analysis, documentation reports, a detailed design skeleton that an implementor can use, a test plan outline that a tester can use, a change history that a maintainer can use, and a project history that can track the design progress.

At the design stage there is much more room for analysis tools than at the requirements stage. One aspect of an Ada design that must be carefully analyzed at the design stage before much effort is spent in implementation is the package organization. In Ada, when a package is recompiled, all the programs using that package must be recompiled. Ideally, the module-package relationships will be simple, without a great deal of interdependencies between them. Tools showing the dependencies between packages and compilation units will be very useful at the design stage to minimize compilations when changes are made and to minimize communications between packages and multiple modules. Whether the design is written in syntactically correct Ada or in a mixture of legal Ada and prose that uses Ada keywords, the design tools should be able to automatically generate a syntactically legal Ada program turning the prose into comments. This program skeleton, which is automatically generated from the design description as the design skeleton was generated from the requirements, cannot help but be traceable to and consistent with the design.

TESTING SUPPORT

In addition to the debugging facilities provided by the compiler, there are a number of formal testing techniques that the environment should support with test tools. Formal testing should be supported at both the single-module and system (after integration) levels. The environment should maintain a record of the test description, test data sets used, modules tested, and test results. The testing information is useful not only in accounting for the test performed but also in determining the retesting requirements for the maintenance and enhancement phase. In formal testing there is the need to generate appropriate tests to demonstrate that the software performs correctly. There are several approaches to testing, each of which can be supported by a test tool that assists in test data generation. The test data generation tools provide data sets for exercising the software in a particular way. Tools performing boundary testing, symbolic execution of loop constructs, checking of assertions from a requirements standpoint, stress testing, and path testing should be provided.

Test harnesses to assist the user in exercising single or multiple modules in a simulated system environment should be easily fabricated. A general test harness that provides hooks to program-defined data can save effort and result in more thorough test cases. Tools should also evaluate testing thoroughness. Criteria that can be used are

1. Showing that the complete input space for small modules is exercised
2. Deriving the response function for a module to compare it to the sampled ideal function
3. Checking that all combinations of paths for a small module, and all path segments for a program, have been exercised
4. Demonstrating that all functional requirement paragraphs have test cases that have been used

For the purpose of demonstrating that optimal performance has been achieved, the places where most of the time is being spent should be determined. It has been shown that only 10% of most code needs to be optimized for maximum performance. Test tools should identify these areas for possible redesign.

SOFTWARE MAINTENANCE AND ENHANCEMENT

Pennington[4] has estimated that software maintenance consumes about 60% to 85% of the total software life cycle costs. Since maintenance is the most expensive part of computer software, the environment must provide maximum support for the maintenance effort. The tools discussed in preceding sections will help generate high-quality software and documentation. These tools will reduce many of the current software maintenance problems. However, even the most reliable software can contain errors, and all software is eventually modified.

Software maintenance can be divided into two categories:

1. Correction of faulty programs
2. Modification or enhancement of processing capabilities

To aid in the correction of faulty programs, tools will be required for establishing a test environment to check reported errors. Diagnostic facilities will also be necessary to aid in tracking error symptoms to their source.

Corrected software must be retested to ensure that no new errors were introduced. The test conditions that demonstrated the error must be incorporated into the test plan. The changes must be documented, and change notices must be distributed.

Major changes to a software product, whether corrections, modifications, or enhancements, will require additional support from software tools. Changes in the user's software requirements can necessitate dramatic changes in the design of existing code. Tools to provide assistance for these *worst case* conditions will be required in any comprehensive software support environment.

The appropriate place to start making changes in computer software is with the statement of the user's requirements and the specifications for the software product. These documents describe the user's needs and how the software is to support those needs. It is, therefore, most important to keep this documentation up to date to direct the modification effort and to allow for future maintenance efforts. Requirements and specification documents in machine-readable form can be easily reviewed and edited to reflect any changes to be made.

Machine-readable software specifications can be expressed in forms that can be analyzed automatically for completeness and consistency. Therefore, changes to the specifications can be checked for conflicts and missing information. Identifying such errors at the earliest possible time minimizes the cost of their correction. Automated analysis of specifications can also produce a skeletal design that aids in evaluating the extent of design changes precipitated by new requirements.

The aids described for generating test data and providing the necessary testing environment (test harness) will find use in the retesting phases. It is rarely sufficient to test just the changed part of a program. A complete check of all processing capabilities is necessary to verify that no adverse side effects from any modifications affect other parts of the program. Modifications that are necessary to correct faulty software may indicate a shortcoming in previous testing efforts. The criteria for acceptable test thoroughness can be reevaluated and changed to improve testing reliability. Performance improvements can be measured and verified during retesting.

Documentation of the testing activity, including descriptions of all tests and a log of all tests conducted, will be assisted by the testing tools. An inventory of all tests will be maintained, as well as a history of the tests performed on all software. Test histories will be maintained for individual-unit tests, subsystem and system level integration tests, and final system checkout.

The final item of responsibility of software maintenance is to keep track of various versions of modified programs. The software maintenance environment will aid in documenting program versions and storing distribution information such as release dates and names of recipients. The documentation will include installation instructions, user manuals, and system descriptions that can be compiled from the project database.

MANAGEMENT TOOLS

Management activities can be broken down into five basic categories:

1. *Planning*—preparation of schedules, budgets, resource estimates and other factors relevant to the execution of a project

2. *Staffing*—assignment of personnel to organizational positions
3. *Controlling*—enforcement of directives and management decisions
4. *Directing*—providing direction to project personnel to support management objectives
5. *Organizing*—establishment of the project structure, positions, and lines of authority

These activities are pervasive in software development, being necessary during all phases of the software life cycle. Up-to-date information regarding the project status and the staff activities must be available for effective management. The environment can play a large role in providing the necessary information in a timely fashion and in a format suited to someone removed from the technical aspects of the project.

The common project database will play a large role in supplying the needed data to the manager. The environment can automatically and accurately record as complete a history of the project's development as management needs. This information can be reported to the manager, who plays a relatively passive role in the building of the database, by a series of management tools.

For planning activities, one of the most important reports would be one indicating the degree of completeness of the project. At the beginning of the project, the manager could enter into the database a proposed schedule of events, including recognizable milestones. The schedule could be updated as the project progresses and unforeseen events occur; however, it would also present the means for the manager to note which project tasks are falling behind schedule and which are going as expected. Exception reports can indicate trouble spots requiring special attention.

The manager should also be able to see at any time how well the actual figures match the projected costs by entering budgetary information into the database. The system can automatically record the computer resources used to date, together with a record of personnel active on the project. From this information, status reports on expenditures and indications of possible trouble spots where projections are far off can be automatically generated for the manager.

Once planning information is entered, special tools can analyze the schedule of tasks to determine those critical paths whose successful timely completion is vital to the overall successful completion of the project. Interactive scheduling programs can assist the manager in developing a schedule that tries to optimize staff and material resources. When a project runs into trouble and requires rescheduling, such programs can suggest new schedules consistent with the revised information.

Another aspect of control is accountability. The environment enhances accountability because all activity within it is recorded. If a module is erased from a library, the environment will indicate who issued the erasure command. If a document is modified, the environment will be able to report who the editor was. With this automatic recording and reporting capability, programmers will also be discouraged from mischief since their identity will likely be known.

SUMMARY

Future Ada environments will support the entire life cycle of a system, from helping to budget personnel to helping with software maintenance. The code development tools presently being implemented will be only a small part of future Ada environments.

ACKNOWLEDGMENTS

Many present and past GRC personnel have contributed to the ideas contained in this paper. In particular, Carolyn Gannon, Nancy Brooks, Reg Meeson, and Art Pyster made substantial contributions.

REFERENCES

1. Ross, D.T., and K.E. Schoman, Jr. "Structured Analysis for Requirements Definition." *IEEE Transactions on Software Engineering,* SE-3 (1977), pp. 6–15.
2. Alford, M.W. "A Requirement Engineering Methodology for Real Time Processing Requirements." *IEEE Transactions on Software Engineering,* SE-2 (1977), pp. 60–69.
3. Pierce, R.A. "A Requirements Tracing Tool." *Proceedings of the Software Quality and Assurance Workshop,* November 1978, pp. 53–60.
4. Pennington, R.H. "Software Development and Maintenance—Where are We?" *Proceedings of the IEEE Computer Software and Applications Conference,* October 1980, pp. 419–422.

Stepwise structuring: A style of life for flexible software

by ERIK SANDEWALL, STURE HÄGGLUND, CHRISTIAN GUSTAFSSON,
LENNAT JONESJÖ, *and* OLA STRÖMFORS
Linköping University
Linköping, Sweden

ABSTRACT

In a life cycle perspective on software, the paper describes a strategy for initialization and successive growth of software, which emphasizes flexible introduction and flexible use. The examples in the paper are taken from office information systems or personalized data processing systems.

The key points in the paper are as follows:

1. The system should be organized so that it allows multiple representations of the same information, particularly as images *(bitmaps)*, text, and structured data.
2. New applications should first be started by using representations with relatively little structure (such as images) and only gradually shift to using more structured representations.
3. It is valuable for the end user to be able to control and make use of the gradual introduction of more structure.
4. It is useful to have software tools that facilitate the interactive work of introducing more structure into the information. Some tools that have been implemented in this project are described.

A PRACTICAL CASE: FROM TEXT FILE TO STRUCTURED DATA

When we have used office information systems in our own work, we have repeatedly found it useful to use plain text files as an interim representation before building a conventional data file. Consider a very simple example: an international address register, which contains name and address information correctly in the various formats used in different countries and which is used for one single purpose: printing adhesive mailing labels to be put on envelopes. If this address register is represented as a text file, it can be edited (with a standard text editor) and it can generate the required labels (using the standard PRINT operation).

Such an implementation leaves with the user, of course, the responsibility of making sure that all entries are correct—e.g., have the correct number of lines and observe the maximum line length. However, it is not difficult for the user to understand these requirements. If there are any mistakes, they can be observed when the labels are printed out, and the user can readily solve the problem. This arrangement has the fundamental advantage that *the user can easily master the system.* For the user, that is an advantage that is often worth the price of extra attention.

The text-editor implementation becomes impractical, of course, when the number of addresses in the directory increases and when the same information is to be used for multiple purposes. The application is then converted to a file of records in the obvious way—to a structured representation. The extra effort of converting the existing text files to the record format can be avoided if one implements the structured representation right from the start. However, it can be worth the effort, since the first stage, using the text file representation, provided a body of experience of several kinds: experience of all the odd varieties of addresses that may occur in practice (which is useful for the implementor), a familiarity with the computer system as a tool (which is significant for the end user), and finally a check of possible practical problems with computer-based solutions (such as mechanical problems with the printer and the labels).

This simple case examplifies the first two of the general principles that we proposed:

1. The data that are contained in an information processing system occupy a slot in a spectrum from less structured (in our example, the text file) to more structured (in our example, the file of records).
2. It is useful to let the early stages of software development be based on less structured data and to increase the strength of structure as the system matures. A significant advantage of this approach is that it is conservative with respect to structuring; i.e., one does not introduce more structure than necessary for processing. The organizational effects of different information media with different levels of structuring has been studied by Innis[1] and Taylor.[2]

In the remainder of the paper we shall argue that there are more than two significant points along that spectrum and discuss their character. We shall also argue an additional point:

3. Software systems for interactive information processing should support more than a single point on the structuring spectrum. In other words, data with different levels of structure should be able to coexist in the same system.

Finally, we refer to systems that have been implemented and used in our laboratory that have allowed us to develop and test these principles.

SOME OTHER EXAMPLES

Let us now discuss additional examples of applications whose data have a place along the spectrum from less structure to more structure.

References in a Text

A good example from the academic environment is the preparation of the bibliography for a scientific paper. At first the manuscripts are just text files. The first transformation is to factor out the references individually as small text files (*notices,* using the term of Sandewall et al., 1980.[3]) The main text file is changed so that it contains only the expansion command, with the file names of the text files for the various references as arguments. A preprocessor must then be used before the regular text formatter in order to reinsert the small text segments for each reference.

Even this first step is valuable because it makes it convenient for various papers to share references. A second step may be to change the text file representation of each reference to a structured representation as a record in a database. Again in this example, it is valuable to have a body of practical data at work in the intended application before the structures are decided.

From Image to Text File

We return to the problem of mixing paper-based and computer-based documents and consider an information workplace, i.e., an office or another working environment where a large number of documents are processed by people. By tradition, those documents arrive on paper.

It would be impractical to key all the contents of those documents into the computer system—i.e., to convert them to the form that we have called *text* in the previous section. Not only is it expensive to do the keypunching; it is also difficult to support all kinds of figures, tables, photographs, etc.

The obvious solution, by analogy with the argument in the previous section, is to recognize the original *image* of the arriving papers as another representation (probably implemented by raster-scan techniques), along with text and structured data. The image is of course less structured than the text: The conversion from text to image is done automatically by formatters and printout devices; the conversion from image to text is usually done manually (by key-typing), and only in some cases automatically.

The support of the image representation requires hardware as well as software. The following is a scenario for what the system could be like. The personal work station consists of a keyboard, a text screen (i.e., a conventional character display terminal), and an image screen (e.g., a full video screen). The direct user-computer dialogue is performed using the keyboard and the text screen—e.g., for issuing commands to the system. The system also contains a long-term memory for image data, using either photographic or electronic storage technology, and a short-term memory for the same kind of data (e.g., in digitized form on a disk memory).

Both kinds of image memory are kept at a central location and may be viewed from all work stations. It is well known that the technology that makes that possible practically is becoming available. Information of lasting value, such as printed reports (or reports that used to be printed), are stored in the long-term memory. In a research setting, this would also include, for example, scientific journals. Information that has just arrived and that is of general interest to the user community (bulletin board information and circulation list information) goes into short-term image memory and is transferred periodically to cheaper long-term memory. Newspaper clippings, advertisements for new products, and (in a research setting) calls for papers for conferences are examples of information that could be handled in this way.

From the perspective of the user, paper-based information that arrives in one copy to the organization is available immediately to everybody on the image screen in his or her office.

Again, the advantages of the strategy should be fairly obvious: Our everyday office life includes many documents that are, properly speaking, images, and that cannot be easily expressed as text without significant loss of information and readability. An electronic office system that is able to represent image and text side by side makes it possible to shift from one representation to the other exactly when it is worthwhile.

Redundancy in Structured Data

There is also a later step, after the conversion from text to structured data, which may be either a *normal form* representation, in the sense of database theory, or a so-called *truth maintenance system,* in the sense of artificial intelligence. If the textual stage in the conversion chain is interpreted to contain all the texts that are required by the organization and if the structured-data stage deals with structured forms of the same texts, then the next stage again should be one where the user is relieved of the duty of maintaining redundant information in cases where the same information is used in several texts. This can be achieved either by reducing the structured data to a normal form, with the standard techniques; or by keeping redundant information in the system together with operators that automatically maintain the consistency, which is what truth maintenance systems do.

The advantage of the normal-form approach is, among others, that it can store large amounts of data economically. An advantage of the truth-maintenance approach is that it can be more concretely understood by the user: the machine still contains the user's documents, and there are "demons" that propagate new information to all the relevant places. Those demons can be created and removed at will and can also be designed so that they can be asked about the reasons for their actions.

A LARGER CASE STUDY: PERSONAL PLANNING INFORMATION

Several experimental office information systems provide facilities intended to facilitate the users' personal planning: calendars, agenda lists (such as "to do" lists and tickler files), and others. Such facilities can serve a widely perceived need when they make it easier to find common time for meetings and appointments. (The potential disadvantages of making it even easier to fill up people's entire days with meetings have not been discussed as much.) Further work along these lines is envisioned: Morgan[4] points out that "in true automation, the control of when to use the tools is placed in the machine support system," and he suggests that that is a desirable goal: "Similar work at Xerox, IBM, and MIT holds much promise for truly automating in the office environment" (p. 785).

Although formal reports are hard to find, informal evidence suggests that computer-based personal planning systems do not usually become popular. We believe that the following factors contribute:

1. Many people want to be in control of how they use their time.
2. Personal planning information is needed the most by people who move around a lot. At least with today's technology, a computer terminal is not available when and where a decision is made to update a plan.
3. The computer-based system cannot compete, in terms of overall convenience, with the paper-based system of handwritten notes.[5]

This does not mean that all is well the way things are usually done. The available range of literature, courses, and tools for personal planning suggests that many people are not satisfied with how they use their own time. Some of those tools could be computer based.

Clearly, personal planning information has a structure. Some tasks such as meetings occupy a fixed location in time; others are limited by deadlines, or by requirements that things be done in a certain order. There is also a goal structure, since most tasks are intended to serve a purpose. Finally, many tasks have other information attached to them: the task of calling a person can be executed only if a phone number is available; the task of traveling to another city is associated with the information that goes into the travel expense form.

At least for a computer professional, it is tempting to diagnose that problems arise because the structure of the information is not made explicit and to implement a piece of software that will administer the information, properly structured. This would be another example of going from a less structured, paper-borne representation of the information (often implemented as a heap of paper slips, with notes scribbled on them), directly to a more structured, computer-borne representation. According to the principle that we argue in this paper, one should not attempt to do that.

In this case, the intermediary station is not computer-borne texts, but instead paper-borne structures. The following description should be interpreted as an example of what one could do, rather than as a specification. The paper-based tool, which represents many of the structures in personal planning, could be a small, looseleaf binder, with tab sheets organizing the papers in the binder into sections and subsections. There could be sections for personal calendar-style time planning (on several levels of time scale); for the schedule of the whole organization, which serves as background for the personal schedule; for agendas and deadline-directed tasks; and for the various kinds of information that are attached to the tasks and sometimes prerequisites for performing them.

Furthermore, the structure provided by the tab sheets should be further refined by a repertoire of different forms used in the binder. It is natural to have special forms for calendar sheets, address directory sheets, and agenda sheets; and the looseleaf structure would allow new forms to be introduced as significant new structures are recognized.

A structured, paper-based planning tool of this kind serves a purpose in itself, and such systems exist already in the office supply market. They are relevant to the topic of the present paper, because we argue that such a paper-based planning tool is necessary before a computer-based tool can become worthwhile. Provided that the integrity of decision making is preserved, the individual user may find it beneficial to arrange that the information in his or her planning book interacts with the information in the computer.

Interaction means that information indeed goes both ways. For example, it is clearly convenient to let the address/telephone directory in the planning book be a selective printout from a file that is shared in the organization, but it is also natural to treat the handwritten updates in one person's address/telephone printout as a source of update information for the database. Thus the interaction between paper-borne and computer-borne information should be viewed as a *paper refresh:* The user brings in a set of paper sheets with handwritten corrections, updates the information in the computer accordingly (or obtains assistance for that chore), and receives a clean set of printouts confirming that the updates have been performed.

The paper refresh operation is of course similar to how programmers work with listings of programs. It may also serve as a model for how other items of personal planning information, such as weekly schedules, communicate with the computer.

The structuring of personal planning information will then have proceeded top-down, and differently from the bottom-up structuring that we discussed above for the publication referencing. Top-down structuring aims at providing an overall structure, within which yet unstructured parts may continue to exist. For example, in an information system which supports image information in short-term memory as described above, it may be sufficient to store an image of each person's weekly plans, so that it is available for others to watch. (Dividing the plan into two columns, one of which is not publicly visible, is a natural modification.) With that design, no software can inspect or modify the contents of the week's plan—a limitation that many users will consider a distinct advantage.

SOFTWARE DEVELOPMENT STRATEGIES

We described initially how the various representations may be successive stages in the system's development process. One starts with a less structured representation and later shifts to a more structured representation when the time is ripe. The body of available information at the time is converted to the more structured form, and there is some procedure (e.g., a formatter or a report generator) that is able to recreate the less structured form from the more structured one. The more structured representation becomes the source; i.e., it is henceforth the object of successive editing.

In some cases the user may wish to keep both representations permanently. For example, ordinary business cards contain some information that it may be worthwhile to change to a more structured form in the address directory, but also some other information which is best kept as it is, such as the logotype of the company or perhaps handwritten notes on the card. The user may keep both the image of the card and the database entry in his or her OIS in such a way that one can easily go from one to the other. In this case it is unclear which of the representations should be thought of as source in the above sense.

An additional and more sophisticated case occurs when the choices of representation are used alternatingly. In an application it is often easy to find a more structured representation that accounts for most but not all of the cases. Returning to the example of the reference list in the scientific paper, most quotations fit into one of a small number of cases (book, paper in a journal, internal report, etc.); but there are also occasional references that do not fit those patterns. In the transfer from the textual representation of the individual reference to the structured representation, the user might then elect to retain the textual representation for the odd cases.

This easy way out has two drawbacks: Search operations (e.g., the search for papers with a certain author) will often not "see" the odd cases, and transformations (e.g., alternative formats for presentation of the quotation, with italics for the title of the paper, the journal, etc.) have to be done manually for the odd cases. However, those drawbacks may be easily acceptable if the volume of information is moderate and if the system is always used interactively, as is often the case in OIS. It is much worse to have to think in advance of all the cases that may possibly arise—or to deal with an inflexible system where some of the cases that should have been thought of in advance have not been.

Two general observations are that one and the same system should be able to account for the various representations of information, and that each user should be able to understand how those representations for the same information can be used and exchanged.

Comparisons with Other System Development Methods

Conventionally, software engineering has recommended a sequence of carefully separated steps (often illustrated as a staircase or waterfall) from specification of needs to operation and maintenance. Stepwise structuring recommends instead that that an initial system should be put into operation early, using general-purpose software that is able to support low levels of structuring; and that only as more experience is gained should the level of structure be increased. The disadvantage of the conventional method is, of course, that it is often difficult to understand needs and do the specifications in advance.

The method of *rapid prototyping* has been proposed repeatedly as another way of dealing with that phenomenon. It has often been quoted as a raison d'etre for various incremental programming languages, such as CS4,[6] Lisp,[7] or APL.[8] However, prototyping cannot deal with the fact that users' needs change continuously during the system's lifetime (and in fact, that the system's lifespan is often limited by its ability to adapt to these changing needs). Stepwise structuring does address that issue; but at the same time it will clearly require other kinds of software tools in order to be practical—for example, tools that support the transition to higher structuring levels.

The method of *structured growth* has been proposed by one of the present authors[7] as a strategy for the gradual extension of software. The idea there is to build an initial software system with relatively few facilities, but with an organization that supports the gradual incorporation of more and more features. Thus it is closer to stepwise structuring in character; but, whereas the method of structured growth emphasizes the gradual accumulation of more software and more variants of data structures, stepwise structuring emphasizes transformation of data between structure levels as the most significant event during the development process.

SOFTWARE TOOLS FOR STEPWISE STRUCTURING

The method of stepwise structuring formulated in this paper immediately suggests the need for a number of software tools:

1. *Support for mixed data representations.* The most important tool is an information management system (IMS) (a kind of editor) that is able to handle several kinds of data at the same time—e.g., text, structured data, figures, and images. If some data are kept on a lower structuring level, even after the bulk of the data has been transformed to a stronger structure, then this IMS will be the working tool of the computer user. But even if complete transformations are done at one time and all data are thereafter in the stronger structured form, the mixed-structure information management system is a necessary tool for those who do the conversion work.
2. *Data parsers.* Regularities of data often arise spontaneously within one structuring level; therefore a data parser can be used as a tool for increasing the structure level. For example, an address directory has fairly regular contents, even if it is stored as a text file. But of course one cannot assume that all data will fit into the presumed syntax. A data parser that is going to be used for this purpose must therefore be embedded within an IMS as just described so that it can be run under strong user control.
3. *Catalogs.* One aspect of such an IMS for mixed data is that it must include the services of a conventional file directory. The use of directories for text files is universally understood, but when one starts to use a very large number of small text files, the function of the dictionary changes from being a way of assigning mnemonic names to individual files to being a database where combinations of named objects (*entities,* in database jargon; *concepts,* in semantic network jargon) from the application domain, have *text objects* (small text files, big strings) associated with them.

Similarly, the image representation of information is practical only if it is annotated by catalogs, which, for example, will identify where a certain issue of a certain journal is stored or in which picture frames a certain article (defined by author, title, etc.) occurs, or where a certain quotation within a certain article is located. In our example, the catalog for the image information tends to merge with the database used for generating references in the bibliography in new papers. That example illustrates a general principle: We really do not need a system that is just a catalog; we need a database that is organized in terms of concepts from the application domain and that *among other things* contains what used to be catalog or directory information.

When a text or an image is annotated in a catalog or a database, some of its structure is already being identified. This suggests that the structuring operation (from image to text and from text to structured data) is often a top-down process, where one first identifies the top-level structure and decomposes the original image (or text) to a number of smaller images (or texts), which may then be again decomposed. Several of the examples follow this pattern, and a catalog in our wider sense can be viewed as the software support for top-down structuring.

There are also many examples of bottom-up structuring, where small parts of a text are broken out, but the whole text retains its character as text. The above example of the references in the research paper is a case in point, and one can add the examples of yearly activities reports, as well as curricula vitae, where lists of similar events (seminars, travel, etc.) can be structured in a bottom-up fashion.

4. *Reconversion tools.* These last examples remind us of another useful, general-purpose tool: a preprocessor for the text formatter, which takes a source-source file as input, recognizes the macro expansion commands in that file, and produces as output a source file, which may be input to the regular text formatter. For the bibliography example, the user defines a command that fetches a text file with a given name and embeds it in the main file. Later, when the individual references are upgraded from text to records, the command is redefined so that it fetches the same information from the database and annotates it appropriately (e.g., using font shifts).

   In general, there must always be a tool for reconverting data from a higher structuring level to a lower one previously used. As long as a certain structuring level is being used, one is likely to build up a number of services that make use of that level. Services that cannot be substantially improved by using the increased structure can continue to be used as they are if information with the new, higher structure is reconverted to the previously used level as a preprocessor to the service.
5. *Recognition mechanisms.* We have discussed examples where the conversion from text to structured data is done at one time. It is also frequently necessary to identify pieces of data in an incoming text in order to relate it to structured data that are already in the system—e.g., when a computer mail message contains the date and time of a forthcoming event that has to be related to the contents of the user's calendar. In general, one needs software that can recognize structure in surviving lower-level data to such an extent that they can be related to the right point in the existing structure in the system. Kofer[9] describes plans for the design of an interface between the two representations that would be adequate for this purpose.
6. *Prototype implementations.* The contents of the present paper are conclusions that we have drawn from earlier implementation efforts, namely the Linköping Office Information System (LOIS)[10,11] and the ED3 structure editor,[12] which is a candidate tool for conversion from text to structured data.

   One view of ED3 is that it is an editor for tree-structured documents, where the structure may be the one of chapters and sections inside one document, or the dictionary structure that organizes a collection of documents, or both. In an editing session, the user starts at the root of a tree and is offered a repertoire of operations for navigating and modifying the tree structure. The leaves of the tree are pieces of text and are modified by a cooperating text editor.

   ED3 has more recently been extended with support for leaves that have other types than text, particularly vector graphics and tables. As such, it illustrates the required characteristics of a dictionary that encompasses several representations, as discussed earlier in this section.

   But another view of ED3 is that it simply maintains a conventional text file together with a bracketing structure that points out the beginning and end positions in the text of blocks that may be nested recursively. When the user views a position in the ED3 tree, he/she views one selected block in the text file. The surrounding blocks are not seen at all, and in the contained blocks only the first line is seen.

   This view of ED3 is actually closer to the actual implementation. It also explains how ED3 may be useful as a tool for the transition from text to structured data: It contains commands whereby the user can conveniently bracket the text file into recursively nested blocks.

   In another project we have developed the Carousel system[11] which shows how a hierarchical information structure, similar to the one used in ED3, can be the basis of a very concise system for many of the basic services in an office information system, such as forms management and command-oriented user dialogue.

Finally, an extensible preprocessor for the formatter has been implemented within Interlisp and has been applied to a number of different uses, including the administration of reference lists. It was used for the preparation of this paper.

Work in progress includes the formal specification of an IMS that, among other things, should be a good software support environment for application development by stepwise structuring.

ACKNOWLEDGMENTS

This research was supported by the Swedish Board of Technical Development under Contract Dnr 80-3918.

REFERENCES

1. Innis, Harold A. *Empire and Communications.* Oxford: Oxford University Press, 1950.
2. Taylor, James R. "New Perspectives on the Office of the Future." In *Proceedings of the International Workshop on Office Information Systems.* Paris: INRIA, 1981.
3. Sandewall, Erik, Göran Hektor, Anders Ström, Claes Strömberg, Ola Strömfors, Henrik Sörensen, and Jaak Urmi. "Provisions for Flexibility in the Linköping Office Information System (LOIS)." *AFIPS, Proceedings of the National Computer Conference* (Vol. 49), 1980, pp. 569–577.
4. Morgan, Howard Lee. "Research and Practice in Office Automation." Invited paper. S. H. Lavington (ed), *Information Processing 80.* North-Holland, 1980.
5. Maryanski, Fred. "Guest Editor's Introduction." *Computer,* 14 (1981), p. 11.
6. Berild, Stig, and Sam Nachmens. "CS4—A Tool for Database Design by Infological Simulation." In *Proceedings of Third International Conference on Very Large Data Bases.* Published in 1977; available from IEEE Computer Society, Long Beach, California.
7. Erik Sandewall: *Programming in the Interactive Environment: The 'Lisp' Experience.* ACM Computing Surveys, Vol. 10, No. 1, pp. 35–72, March 1978.

8. Gomaa, Hassan, and Douglas B. H. Scott. "Prototyping as a Tool in the Specification of User Requirements." In *Proceedings of the 5th International Conference on Software Engineering.* New York: IEEE, 1981.
9. Kofer, G. Reinhard. *Some Software Integration Technology Concepts for Saving Money While Doing Empirical User Research.* In *Proceedings of the International Workshop on Office Information Systems.* Paris: INRIA, 1981.
10. Hägglund, Sture; other authors (names not given in report). "80-talets elektroniska kontor. Erfarenheter från LOIS-projektet." ("The Electronic Office of the 80's. Experience from the LOIS Project.") Research report LiTH-MAT-R-81-4, Software Systems Research Center, Linköping University, Sweden, 1981.
11. Sandewall, Erik. "Unified Dialogue Management in the Carousel System." In *Proceedings of the SIGACT/SIGPLAN Conference on the Principles of Programming Languages.* Albuquerque: 1982.
12. Strömfors, Ola, and Lennart Jonesjö. "The Implementation and Experience of a Structure-Oriented Text Editor." In *Proceedings of ACM SIGPLAN/SIGOA Symposium on Text Manipulation.* New York: Association for Computing Machinery, 1981.

HITS: A symbolic testing and debugging system for multilingual microcomputer software

by TAKESHI CHUSHO,
ATSUSHI TANAKA, and
ERI OKAMOTO
Hitachi, Ltd.
Kawasaki, Japan
and
AKINORI HONDA
and TORU KUROSAKI
Hitachi, Ltd.
Yokohama, Japan

ABSTRACT

The use of a large-scale computer is the key to the development of increasingly numerous and large-scale microcomputer software programs. HITS (Highly Interactive Testing-and-debugging System) constructs an integrated programming environment for 68000 microcomputer systems on a large-scale computer in cooperation with language translators. This system supports efficient and effective software validation from module testing through system testing. Functions of HITS are provided in the test-procedure description language, in which test data, expected results and the testing environment are described and separated from the target program. The main features are (1) symbolic support of both a high-level language and an assembly language, (2) module testing facilities such as driver and stub definitions, (3) a testing coverage monitor for branch testing, (4) debugging commands added temporarily to a test procedure from a terminal, and (5) a macro definition for language extension. HITS has already been used at many sites. In our early experience of applying it to the software development of various communication systems, software productivity and reliability were considerably improved.

INTRODUCTION

Software development for microcomputer systems is entering a critical stage. This is because the programming environment is still poor, even though microcomputers have been applied extensively to various fields, many of which have required high reliability. Furthermore, large-scale software has begun to be developed as 16-bit microcomputers have come into wider use. For example, we have developed 100 ~ 200 kilo steps of software for a digital switching system using 68000 microcomputers.

To date, almost all programming environments for microcomputer-software development have been constructed on the target microcomputer or on a development support system in which a microcomputer is embedded. Such resident support systems, however, provide limited facilities. That is, the programming language is usually assembly language. Furthermore, a debugger supports only dump, patch, breakpoint, and trace on a machine-language level. There are no operating systems with various useful utilities and powerful file management for software development as there are in a large-scale computer.

There are two effective solutions to these problems:

1. Programming in a high-level language and testing and debugging on a source-program level.
2. Using a large-scale computer for developing software, from programming through validation.

These solutions have been partially adopted in previous studies. For example, a high-level language, PL/M, for Intel's microcomputer families, was early developed. However, the software development system is not sufficient for software validation, because it mainly supports debugging, not testing such as the symbolic description of a test procedure.[1] Another example is the microcomputer software engineering facility, MSEF, which uses a minicomputer.[2] Although this system is aimed at supporting a wide range of microcomputer-software development, the testing facility is limited to management of the relationship between a target program, its input data, and results under the hierarchical file system.

We have incorporated both of these solutions in an attempt to deal with the problem of developing large-scale software for digital switching systems. First, a system description language for microcomputers, S-PL/H, has been developed and its cross compiler has been available in the Hitachi M-series computer system since the end of 1980.[3] S-PL/H is a superset language of PL/M and it provides both the basic facilities of PL/I and microcomputer-oriented facilities.

Next, a testing and debugging system for microcomputer 68000 software, HITS, has been developed for efficient and effective software validation using the large-scale computer.[4] This has been available since the spring of 1982. HITS constructs an integrated programming environment for microcomputer software development in cooperation with S-PL/H.

The main requirements for HITS are a wide range of supports for various aspects as follows:

1. Support ranging from small-scale software through large-scale software,
2. Target programs in both a high-level language and assembly language,
3. Testing facilities ranging from module testing through system testing,
4. Compatibility of testing facilities and debugging facilities,
5. Executions in interactive mode and batch mode.

This paper describes the design concepts and functions of HITS and some application results.

DESIGN CONCEPTS

Many different techniques and tools for software validation have been developed, such as data-flow analysis for automatic error detection, symbolic execution for automatic test-data selection, and assertions for correctness proof.[5] Many of them, however, are not practical for large-scale software validation because they require enormous computing resources.

Therefore, we still must depend on "exhaustive testing" in which a lot of data are evaluated against the corresponding expected results. Our goal is to improve the efficiency and effectiveness of such dynamic testing. HITS was thus developed on the basis of the following design concepts:

1. *Environment:* use of a large-scale computer
2. *Coverage:* support of module testing, integration testing, and system testing
3. *Function:* support and unification of systematic testing facilities and interactive debugging facilities
4. *Object:* program modules written in a high-level language and assembly language
5. *Ease of use:* minimization of preparations and operations, such as symbolic commands and a test-procedure library.

First, the use of a large-scale computer provides the following advantages:

- integrated file management for source programs, object programs, test data, test results, and path-coverage data,

- parallel processing of both module testing and integration testing under a time-sharing system.

Figure 1 shows the system configuration of HITS. The second item, systematic testing from module testing through system testing in this configuration, will be described in the next chapter.

The third item is based on the idea that testing and debugging cannot be separated. Of the conventional tools for software validation in practical use, there are many that provide only debugging facilities. The others provide only testing facilities. For example, although MTS[6] and TPL[7] are excellent tools for module testing, they do not support debugging. Furthermore, the former requires much preparation time because of target-language independence. The latter is limited to tests having only Fortran subroutine parameters. In HITS, when an error is detected by the execution of a test procedure that includes test data and the expected results, the test procedure can be executed again interactively while adding temporary commands for debugging.

The fourth item, support of both a high-level language and assembly language, is necessary for the development of system software because assembly language is used for the description of modules requiring device control or critical response time. For example, in the aforementioned digital switching system, 70% of all modules are described in a high-level language, S-PL/H, and 30% in assembly language. Therefore, these two languages are supported so that HITS may be available not only for module testing but also for integration testing and system testing. The fifth item, ease of use, is indispensable to support tools. A test procedure description language for HITS was designed taking this policy into consideration.

SYSTEMATIC TESTING

Software testing is performed in the following steps:

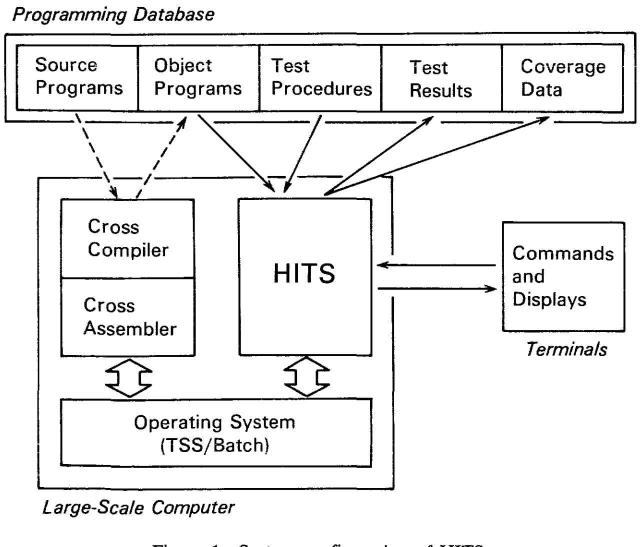


Figure 1—System configuration of HITS

1. module testing for validation of each module function,
2. integration testing for validation of interfaces between related modules,
3. system testing for validation of system function.

To be applied as widely as possible, a test system should systematically and uniformly support all of these steps and not depend on any one particular testing strategy such as bottom-up testing or top-down testing.[8]

HITS provides the following features for systematic testing:

1. All testing steps are supported by providing testing-environment simulation facilities for module and integration testing and a module-binding facility for integration and system testing.
2. Test data can be shared among all testing steps by using a test procedure that includes the test data.
3. Testing-coverage data for effective test-data selection and quality assurance are collected throughout all testing steps.

We will now look at these features in a little more detail.

Module/Integration Testing

Module and integration testing should be performed as thoroughly as possible, considering the following two axioms of productivity and reliability:

1. The later an error is detected, the more it costs to correct it.[9]
2. It is difficult to get a high testing-coverage rate at a later step.[10]

These testing steps, however, require a testing-environment construction that is complicated and troublesome. That is, an upper module, lower modules, global data, and input/output devices for the target module must be simulated. HITS reduces this work with testing-environment support facilities as shown in Figure 2.

Test Procedure

A test procedure includes test data, expected results, and testing-environment simulation, and is described in the test procedure description language that will be discussed later. This procedure is separated from a target module and can be shared throughout all testing steps by eliminating the simulation part, which integration of modules makes unnecessary.

Branch testing

Test-data selection methods are classified into functional testing, based on function specification, and structural testing based on program structure.[11] Branch testing is typical of the latter methods and is supported by HITS.[12] That is, the num-

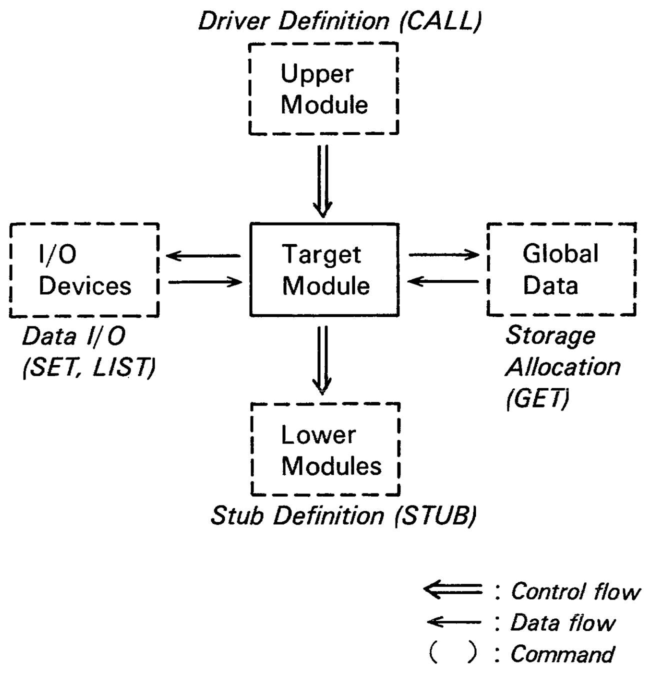


Figure 2—Environment support facilities for module testing

ber of executed branch directions is measured and the unexecuted parts are reported.

FUNCTIONS OF HITS

Test-Procedure Description Language

A test-procedure description language is designed as a command language rather than a procedural language because

1. HITS supports both testing and debugging in a uniform manner, and a command language is very suitable for interactive debugging.
2. Furthermore, a command language is easy to use even for test-procedure description and this satisfies the design policy of minimization of preparations and operations.

The structure of a test procedure is as follows:

```
PROC test-procedure name
  {commands in common use among the following
  test cases}
  CASE the first test-case name
    {commands only for the first test case}
  END
  CASE the second test-case name
    {commands only for the second test case}
  END
  .
  .
  .
END PROC
```

Commands in common use include commands such as those for binding of target modules and storage allocation of external global data. The test procedure is stored in a library and is executed by an EXEC command or is entered directly from a terminal. We would next like to look at command functions.

Simulation of Testing Environment

1. *Driver definition:* Upper module simulation is composed of value assignments to input parameters using SET commands, target-module invocation using a CALL or GO command, and result verification using IF commands. CALL may include value assignments to input parameters.
2. *Stub definition:* Lower module simulation is described in a STUB command whose subcommands may be composed of IF commands for input-parameter checks and SET commands for value assignment to output parameters.
3. *External global data:* Their storage is allocated using GET commands and may be assigned values by SET commands.
4. *Input and output:* Their instruction location is specified as a breakpoint by an AT command whose subcommands are SET commands for value assignments to input variables or LIST commands for display of output values.

Reduction of Test Procedure Description

1. *Macro-definition:* A list of commands used repeatedly is defined as a new extended command by a macro-definition facility. For example, a new command for a result check is defined as follows:

```
CLIST %CHECK
  IF &1 = &2 LIST '<O.K.>' , '&1 = &2'
  IF &1< >&2 LIST '<N.G.>' , '&1< >&2'
END CLIST
```

&n implies the *n*th parameter. Assuming that this macro is used as %CHECK (STATE, 3), if the value of the variable STATE is 3,

```
<O.K.> STATE = 3
```

is displayed. if not,

```
<N.G.> STATE < >3
```

is displayed.

2. *Simplification of object identification:* A QUALIFY command permits references to a local name without qualification that specifies the scope of the name. An EQUATE command replaces a complicated address expression with a new name.
3. *Variation of constant values:* A DATA command defines a sequence of constant values so that a test case can be executed repeatedly while varying only constant values.

4. *Communication among test procedures:* LOAD and SAVE commands permit a test procedure to use data values that are created by another test procedure.

Debugging Facilities

1. *Breakpoint:* The breakpoint is specified by an AT command which may include subcommands executed at the breakpoint. A breakpoint is expressed by the procedure names or statement numbers for the target program in S-PL/H. The specification of the procedure name causes an interruption and requests commands at the beginning of the procedure. The specification of the procedure name following END also functions at the end of the procedure. The statement numbers should be used only for interactive debugging, not for test-procedure description, so that modification of a target program does not cause modification of the test procedure. For the target program in assembly language, a breakpoint is expressed by the label names and hexadecimal offset address.
2. *Trace:* TRACE commands are used for the forward and backward control trace of branches or procedure calls, or for trace of data-value modifications. A BREAK option causes an interruption and requests commands at every trace event.
3. *Debug mode:* The BREAK option also functions at the beginning of the target-program execution if it is so specified before execution of a test procedure. Therefore, at that time, temporary commands for debugging can be entered without rewriting the test procedure in a library.
4. *Off-line output:* A large amount of trace data or dump can be output to a line printer instead of a display terminal by using an OUT option.
5. *Exception handling:* Exception handling can be described in a STUB command with an INTERRUPT option that includes an interruption condition such as an operation-code trap and an address error. References to undefined data are always detected.

DESIGN OF COMMAND LANGUAGE

The command syntax of HITS has the following features in comparison with conventional command languages:

1. procedural concept of block structure,
2. target language dependency,
3. abbreviation of command name.

First, it is desirable that constraints between commands be few. However, when HITS commands are used for description of a test procedure, some commands require subcommands. Therefore, the following seven block structures are introduced:

i. test-procedure block (PROC ~ END)
ii. test-case block (CASE ~ END)
iii. macro-definition block (CLIST ~ END)
iv. linkage block (LINK ~ END)
v. stub block (STUB ~ END)
vi. condition block (IF ~ END)
vii. breakpoint block (AT ~ END).

The last four blocks are used only if they have two or more subcommands. When there is only one subcommand, it is specified at their operands for simplicity. The second feature implies that a user can describe a test procedure on the target source-program level. For example, abstract operands of HITS commands, <instruction-address> and <data-address>, depend on a target language as shown in Table I. Therefore, it is easy to learn and use the command language. The third feature is provided to improve the efficiency of interactive debugging (full names of commands should be used in test procedures for readability). Our abbreviation rule is simple, that is, the latter part of a name can be truncated from an arbitrary position after the first character. If the truncated names of some commands are the same, the system decides which is which in advance, based on the frequency of use.

EXAMPLE

An example is given for explanation of a testing process using HITS. Assume that we develop a program for selecting the maximum of two values that are the minimum values of two groups of values. Two procedures, MINIMAX and MIN, are written in S-PL/H as shown in Figure 3.

A test procedure for integration testing of these procedures is shown in Figure 4, assuming that the lower procedure MAX and a caller of MINIMAX are not written yet. First, two modules, SUB1 and SUB2, including MINIMAX and MIN, respectively, are extracted from a library by an INCLUDE command that is a subcommand of a LINK command. Next, storage for the external global data, X and Y, is allocated. Then, a stub for MAX is defined and several test cases follow.

One of the test cases, C07, is composed of value assignments to global variables, X and Y, invocation for MINIMAX, and result check. The definition of %CHECK has been previously mentioned. Here, two other interesting macro-definitions can be used, namely %PRE and %POST. They are assertions for verification of the precondition and postcondition of a procedure, and are defined as follows:

```
CLIST %PRE
  AT &1 DO
    IF &2 RESUME
    LIST '&1 PRECONDITION: &2 is false.'
  END
END CLIST

CLIST %POST
  AT END &1 DO
    IF &2 RESUME
    LIST '&1 POSTCONDITION: &2 is false.'
  END
END CLIST
```

TABLE I—Command operands differing between S-PL/H and assembly language

Abstract Operand	Details for S-PL/H	Details for an Assembly Language
< instruction-address >	procedure name or statement number	label with offset
< data-address >	variable name	label with offset, indirect addressing, and register indexing

For example, these commands may be inserted before a CALL command in the test case C07 as follows:

```
%PRE (MIN, A(1) > -1)
%POST (MIN, I < 11)
```

The first command verifies that the input parameter A to procedure MIN has at least one valid value. The second verifies that the array variable A was never erroneously referred to out of range in procedure MIN.

When the test case C07 is executed, the following error message is output at %CHECK (RESULT, 2):

```
<N.G.> RESULT < >2
```

An example of interactive debugging for this error is shown in Figure 5. The test case C07 is executed with the debug mode (BREAK option). At the beginning of MIN, the value of the input parameter A is checked. Then, MIN is executed while tracing for modifications of the output parameter B. Finally, the cause of the error in an if statement is detected, and this test procedure is terminated.

```
SUB1: do;
  dcl (X,Y) (10) integer external;
    •
    •

  MINIMAX: proc (var M) public;
    dcl (M,MX,MY) integer;
    call MIN(X,MX);
    call MIN(Y,MY);
    M=MAX(MX,MY);
  end MINIMAX;
end SUB1;

SUB2: do;
  MIN: proc (A,var B) public ;
    dcl A(10) integer;
    dcl (B,I) integer;
    B=A(1);
    I=2;
    do while A(I) >= 0 ;
      if A(I) > B then B=A(I);
      I=I+1;
    end;
  end MIN;
end SUB2;
```

Figure 3—A sample of a target program

```
PROC TP21
  LINK INCLUDE SUB1,SUB2
  GET X,Y
  STUB MAX(P,Q) DO
    SET MAX=P
    IF P < Q SET MAX=Q
  END
  •
  •

  CASE  C07
    SET X=(1,3,5,7,-1)
    SET Y=(2,4,6,-1)
    CALL MINIMAX(RESULT)
    %CHECK (RESULT,2)
  END CASE
  •
  •

END PROC
```

Figure 4—A sample of a test procedure

APPLICATION OF HITS

This system has been released to many factories and laboratories since the spring of 1982. The following advantages of HITS were confirmed.

1. *Writability:* The average number of commands in a test procedure is 4.4 ~ 5.6 per test case for module testing of a digital switching system, although the number of com-

```
{ ready }
OPTION BREAK
EXEC TP21(C07)
{ break at MINIMAX }
AT MIN LIST A
RESUME
{ display and break at MIN }
TRACE DATA(MIN#B)
AT END MIN
RESUME
{ display and break at the end of MIN }
STOP PROC
{ ready }
```

Figure 5—An example of interactive debugging

mands depends on such things as the number of input and output parameters, the number of external data, and similarity among test cases.

2. *Operability:* The target program is automatically tested by entering an EXEC command. This is because various operations required by a conventional debugger are automated or assembled into a test procedure.
3. *Reliability:* The quality of a target program becomes visible with the use of a testing-coverage facility and is improved by adding test cases for unexecuted branches. Reliability of testing is also improved because a test procedure is described on the target-program source level and clearly corresponds to a target program and its testing specifications.
4. *Productivity:* Productivity is improved by the following factors:
   i. early error detection by promotion of module testing,
   ii. high efficiency of test-data generation, execution, and result check
   iii. quick debugging.

   In our experience, when HITS was applied to only module and integration testing, testing cost was reduced by 35% in comparison with the previous testing method using the target computer. For a target program applicable to system testing, testing cost was reduced by 45%.
5. *Maintainability:* It is easy to modify and add test cases because a test procedure is separate from a target program. The test procedure is shared among module, integration, and system testing with only minor changes, and is also available in the maintenance phase of a target program.

CONCLUSIONS

A testing and debugging support system, HITS, for microcomputer 68000 software was developed for efficient and effective software validation using a large-scale computer. The main features of HITS are as follows:

1. All steps of module testing, integration testing, and system testing are supported while sharing test data and accumulating testing-coverage data.
2. Module-testing support facilities for simulation of an upper module, lower modules, external global data, and input/output devices are provided.
3. Test data, expected results, and environment simulation are assembled in a test procedure that is executed under both batch and interactive modes.
4. Both a high level language, S-PL/H, and assembly language are supported on the source-program level.

HITS has already been released to many sites and has improved software productivity and reliability.

ACKNOWLEDGMENTS

The authors wish to express their gratitude to Dr. Takeo Miura for providing the opportunity to conduct this study. They are also indebted to Tan Watanabe, who designed S-PL/H, for his invaluable technical assistance, Mitsuyuki Masui for comments on drafts of the functional specification, and Tatsuro Oishi for modification of the S-PL/H cross-compiler and the cross-assembler that pass symbolic tables to HITS.

REFERENCES

1. *Guide to Intellec Microcomputer Development Systems.* Santa Clara, Calif.: Intel Corporation, 1978.
2. Eanes, R. S., C. K. Hitcon, R. M. Thall, and J. W. Brackett. "An Environment for Producing Well-Engineered Microcomputer Software." *Proceedings of the 4th International Conference on Software Engineering,* 1979, pp. 386–398.
3. *Hitachi Microcomputer System: 68000 Super-PL/H Language Manual.* Tokyo: Hitachi Ltd., 1981.
4. Chusho, T., T. Watanabe, T. Kurosaki, and T. Yamamoto. "Design Concepts of a Microcomputer Software Testing and Debugging System." *The Fall Conference of Information Processing Society of Japan* (in Japanese), 1981, pp. 419–420.
5. Miller, E. F., and W. E. Howden. "Tutorial: Software Testing and Validation Techniques," IEEE Catalog No. EHO 138-8, 1978.
6. *Module Testing System (MTS) Fact Book.* London: Management Systems and Programming Ltd., 1972.
7. Panzl, D. J. "Automatic Software Testing Drivers." *Computer,* 11 (1978), pp. 44–50.
8. Myers, G. I. *The Art of Software Testing.* New York: Wiley-Interscience, 1979.
9. Sorkowitz, A. R. "Certification Testing: A Procedure to Improve the Quality of Software Testing." *Computer,* 12 (1979), pp. 20–24.
10. Holthouse, M. A., and M. J. Hatch. "Experience with Automated Testing Analysis." *Computer,* 12 (1979), pp. 33–36.
11. Howden, W. E. "Applicability of Software Validation Techniques to Scientific Programs." *ACM TOPLAS,* 2 (1980), pp. 307–320.
12. Miller, E. F. "Program Testing: Art Meets Theory." *Computer,* 10 (1977), pp. 42–51.

A global checkpointing model for error recovery

by KRISHNA KANT
Northwestern University
Evanston, Illinois

ABSTRACT

The paper proposes a new concept for providing software fault tolerance in concurrent systems. It combines the traditional global-checkpointing mechanism with the recovery-block concept in order to come up with an easily implementable error-recovery mechanism. This mechanism involves smaller overhead in case of moderate to high process interaction than the schemes considered in the past, which are based upon the idea of local checkpointing. A model for computing the optimum checkpointing interval is also presented. A particular distribution is hypothesized for the coverage of the recovery and the behavior of the model studied in detail for this case.

A. INTRODUCTION

Global checkpointing (GCP) is a popular technique for ensuring that an unexpected system failure does not result in the loss of valuable information. The state of the entire system is saved at periodic intervals (known as checkpoints) so that in the event of a failure the system can be brought to a consistent state simply by resetting it to its last checkpoint. There is an obvious tradeoff between the checkpointing overhead and the amount of computation lost as a result of failure. Many models have been proposed in the literature for computing the optimum checkpointing interval.[2,3]

The fundamental assumption in such a GCP-based approach to fault tolerance is that the system failure is caused by temporary faults such as transient hardware faults, operator error, erroneous input data, timing problems resulting from an unusual combination of circumstances, and so on. In such cases, a rollback and retry procedure would most likely correct the problem. However, if we consider the failures resulting from bugs in the operating software, such a scheme does not suffice. In order to obtain fault tolerance against software bugs, it becomes necessary to incorporate functional redundancy in software. A well-known mechanism for doing this is the recovery-block concept.[7]

As proposed by Randell, the recovery-block (RB) concept uses local checkpointing (LCP), that is, the state saving and restoration is done on a per process basis. If the processes constituting a concurrent program only compete for resources but do not interact otherwise, LCP is clearly the preferred strategy, since the rollback and retry will be limited to only the failed process. However, if we have a system of interacting processes, the erroneous information may propagate from one process to the other before it is detected. This leads to two complications: (1) it is no longer sufficient to maintain only the last checkpoint; in fact, very old checkpoints may need to be kept in order to handle occasional very long rollbacks. (2) the "exact" identification of the points to which the processes of the system need be rolled back becomes extremely complex and costly. (See Kant and Silberschatz[4] and Kim[5] for more on LCP-based recovery in concurrent programs.) In this paper, we explore the possibility of using GCP instead of LCP to reduce the cost and complexity of recovery in concurrent programs.

B. RECOVERY BLOCKS WITH GLOBAL CHECKPOINTING

For this, we simply remove checkpointing from Randell's RB concept; that is, when a process enters an RB, it does not establish any checkpoint. Any failures, including those resulting from the inability to pass the acceptance tests, are handled using the conventional GCP scheme. Thus the checkpoints will be established at periodic intervals and their location would be unrelated to the entry and exit points of the RBs. It is worth mentioning that this use of GCP is not at the level of entire computer system but only at the level of an individual program, which may consist of several interacting processes. In what follows, "system" refers to only one such program.

An important consideration in the use of global checkpoints for error recovery is the coverage, that is, the probability that the rollback will undo the erroneous interactions between the processes. Since the checkpoint interval is fixed in advance rather than deduced from process interaction history, we run the risk of doing too little or too much rollback for a given case. If the rollback is insufficient, then the system will fail again during retry. This situation can only be handled by increasing the rollback span for the next recovery attempt.

We assume that every RB has at least N alternates, where N is the number of previous checkpoints (PCPs) we are willing to maintain throughout the execution of the program. The first $N - 1$ of these PCPs are consecutive (i.e. PCP(1) is the last established, PCP(2) is the one before that, etc.), and the last one, PCP(N), corresponds to the starting system state. The checkpoint to be established next during normal execution will be denoted by NCP. If a failure occurs before an NCP, the goal would obviously be to rollback and restart the system so that the execution proceeds successfully until the establishment of this NCP. If, during reexecution, a failure occurs prior to this NCP, then the rollback span must be increased and execution attempted again as explained above. It is clear that the system must keep track of the location of NCP during retry.

In order to keep track of which alternative of an RB is to be used for execution, we associate a counter ALTNO, initialized to 1 with each RB. We assume that all alternatives of an RB have been designed and coded independently and are approximately equally reliable. Thus we could use them cyclically, that is, if the current version of an RB (identified by ALTNO) is suspected of being faulty, we set ALTNO to mod(ALTNO,N) + 1 for the next retry. The intended recovery algorithm for coping with a failure is as follows:

1. Set a checkpoint counter, K, to 1. K will be used to keep track of how far back the system has been rolled back from the current state in terms of the number of previous checkpoints.
2. Rollback the system to Kth previous checkpoint. Note that if $K = N$, the system will be restarted from its initial

state. For each RB that was entered since the establishment of this checkpoint until the point of failure, set ALTNO to mod(ALTNO,N) + 1. The purpose of this is to make sure that the failed part of the computation is retried using different alternatives. Discard any checkpoints that were established after PCP(K).

3. Restart the system and let it run until one of the following two things happens:
   a. Execution proceeds successfully until the point where NCP should be established. In this case, all computation has been redone successfully and no further action is necessary until the next failure.
   b. A failure occurs before the point for the establishment of NCP. Increment K and go to step 2.

C. ANALYSIS OF GCP MODEL

Our purpose here is to find the optimal checkpoint interval by minimizing the combined cost of checkpointing, recovery, and recomputation. However, an assumption is necessary before this cost could be defined meaningfully. Note that if a rollback to the starting state becomes necessary, an unbounded amount of computation may have to be redone after the rollback. If this recomputation is considered in computing the cost, the cost would be unbounded and no meaningful results can be obtained. Therefore, we consider the costs associated with only the first $(N-1)$ rollbacks. If the probability of requiring rollback to the initial state is extremely small, the results would still be quite accurate and useful. We define the following quantities:

- *cps*: Checkpointing span, that is, the number instructions executed between two successive checkpoints.
- *cpo*: Checkpointing overhead, that is, the number of instructions (or its equivalent in terms of time overhead) required to establish a checkpoint. We assume that the use of a checkpoint in state restoration also involves the same amount of overhead.
- PE_j: Probability of successful execution until the next j checkpoints have been established given that the execution started in a consistent state.
- PR_j: Probability of successful recovery from failure when the rollback span is j.
- PER_j: Probability of successful recovery and retry when a rollback span of j is used. It is easy to see that

$$PER_j = PE_j * PR_j.$$

- *RRC*: Expected cost, in terms of number of instructions, of recovery and retry per failure.
- cost: Fractional cost of checkpointing, recovery and retry.

Let $ncp = cps + cpo$. The overhead of initial rollback and retry is *ncp*. With probability $(1 - PER_1)$, a failure will occur during retry, thereby requiring more severe rollback. The overhead of the second rollback and retry will be exactly $2*ncp$ because PCP(1) must be reestablished during retry. Continuing in this manner, and keeping in mind the assumption stated above, we come up with the following expression for *RRC*:

$$RRC = ncp * \sum_{i=1}^{N-1} i * \prod_{j=0}^{i-1} (1 - PER_j) \qquad \text{where } PER_0 = 0.$$

Let *ier* be the instruction execution rate of the machine. Then the number of checkpoints established per second is (ier/ncp). Therefore,

$$\text{Primary failure rate} = (1 - PE_1)*(ier/ncp) \text{ and}$$

$$\text{Primary checkpointing cost per second} = cpo*(ier/ncp)$$

Thus, the total cost of checkpointing, recovery, and retry (TC) is

$$TC = [cpo + (1 - PE_1)*RRC]*(ier/ncp)$$

whence

$$\text{cost} = TC/ier = cpo/ncp + (1 - PE_1)* \sum_{i=1}^{N-1} i * \prod_{j=0}^{i-1} (1 - PER_j)$$

Now we can minimize cost and compute the optimum *cps*. First, however, we must obtain an expression for PER_j.

C.1 Calculation of PER_j

First note some general properties of PE_j and PR_j. Assuming that the occurrence of software and hardware faults is uncorrelated, PE_j would be the product of PES_j and PEH_j, which are the probabilities that the software and hardware failures do not occur until the establishment of next j checkpoints. Let P be the probability that no failure occurs during the computation performed between successive checkpoints. Then $PES_j = P**j$ assuming that the code used for establishing checkpoints and using them for state restoration is free of software faults. The probability P primarily depends upon three parameters: (a) the checkpoint span *cps*, (b) the "quality" of the software, and (c) its error-detection capability. The last two of these are very difficult to quantify, although several metrics for them have been proposed in the literature.[8] Here we shall simply work with a parameter *ifp*, which is defined to be the probability of failure per instruction of the user code. Then

$$P = (1 - ifp)^{cps}.$$

Note that if fault corrections were taken into account, *ifp* would change as a function of the number of failures experienced. We could use Musa's execution-time model[6] to compute this change. Here we assume that *ifp* is a constant. Thus,

$$PES_j = (1 - ifp)^{j \cdot cps}.$$

PEH_j can be computed using the classical hardware-reliability model. In order to make our analysis independent of the execution speed of the machine, we shall work in terms of number of instructions executed rather than the execution time. We assume a constant hazard rate (in the units per instruction) denoted by λ. Then PEH_j must be an exponential function of the number of instructions executed. Therefore,

$$PEH_j = \exp(-\lambda \cdot ncp \cdot j),$$

and

$$PE_j = PES_j * PEH_j = (1 - ifp)^{j \cdot cps} * \exp(-\lambda \cdot (cps + cpo) \cdot j).$$

Using the fact that *ifp* must be extremely small in any practical case, we have:

$$PE_j = \exp(-j(a \cdot cps + b))$$

where

$$a = \lambda + ifp, \; b = \lambda \cdot cpo.$$

We assume that correct recovery can always be performed by rolling the system back to its starting state. Thus $PR_j = 1$ for $j = N$. It is also clear that $PR_0 = 0$ because no recovery is possible without rollback. Furthermore, PR_j is expected to be a monotonically increasing function of j. We hypothesize that it approaches 1 exponentially as a function of j. Thus

$$PR_j = 1 - \exp(-j/\alpha), \; 0 \le j < N,$$

where α is a control parameter and must be nonnegative. It can be noted that smaller values of α would be desirable. Obviously, α depends upon the extent to which erroneous information can propagate before it is detected. This aspect of the system behavior is primarily controlled by two parameters, (a) the average number of instructions per RB and (b) the extent of interactions between the processes of the system. A reasonable measure for the latter is the fraction of instructions executed by the system that involve or constitute interprocess communication. We denote average RB size by *rbs* and the interprocess communication fraction by *ipcf*. Then α can be written as a function of two arguments, that is, $\alpha(rbs, ipcf)$. We claim the following:

1. $\alpha(rbs, ipcf)$ is very large for $rbs >> cps$ and $0 \le ipcf < 1.0$
2. $\alpha(rbs, ipcf)$ is very small for $rbs << cps$ and $ipcf << 1.0$
3. $\alpha(rbs, ipcf)$ increases monotonically with *rbs* and *ipcf*.

The first claim is based on the fact that no recovery is possible if the average RB size is much larger than the checkpoint span. The second claim is based on the fact that a single rollback would be sufficient for correct recovery if *rbs* is much smaller than *cps* and the interactions between processes are very infrequent. The justification for the third claim should be obvious. We assume that the contribution to α from *rbs* is proportional to the fraction *rbs*/*cps*. The contribution due to *ipcf*, however, is expected to increase with *ipcf* because error propagation due to process interactions generally increases very fast with the level of interaction. We hypothesize that α is of the following form:

$$\alpha(rbs, ipcf) = C * (rbs/cps) + D * ipcf^u$$

where C, D, and u are some positive constants that depend on various characteristics of the system under consideration, such as the quality of acceptance tests, the number of processes, and the complexity of interprocess communication. For example, if the processes show a very complex interaction pattern, the constant u is expected to be rather large, thereby making recovery difficult even when the parameter *ipcf* is fairly small. Since the checkpoint span, *cps*, is the only parameter of interest in the calculation of the optimum checkpoint interval, we can rewrite α as follows:

$$\alpha = c/cps + d$$

where $c = C^* rbs$ and $d = D^* \, ipcf^u$. Let $x = a \cdot cps + b$. Then $\alpha = (a \cdot c - b \cdot d + d \cdot x)/(x - b)$. Also let $y = \exp(-1/\alpha)$. Then,

$$PE_j = \exp(-jx); \; PR_j = 1 - y^j; \; PER_j = \exp(-jx)\,(1 - y^j)$$

$$\text{cost} = cpo/ncp + (1 - \exp(-x) * \sum_{i=1}^{N-1} i * \prod_{j=0}^{i-1} (1 - \exp(-jx\,(1 - y^j)) .$$

The expression for cost can be further simplified as

$$\text{cost} = a \cdot cpo/(x + a \cdot cpo - b) + (1 - \exp(-x)) * f(x)$$

where

$$f(x) = 1 + (1 - (1 - y)\exp(-x) * [2 + (1 - (1 - y^2) \times \exp(-2x) * [3 + \ldots]]$$

C.2 Computation of the Optimal Checkpoint Interval

For $N = 2$, we only need to consider the first term in $f(x)$. Since $x << 1$, we can approximate $\exp(-x)$ by $(1 - x/2)^2$ and compute the value that minimizes cost. The result is:

$$x = (1 - \beta/2) - [(1 - \beta/2)^2 - 2(\sqrt{a \cdot cpo} - \beta)]^{1/2},$$

where $\beta = a \cdot cpo - b$. The model for $N > 2$ can now be solved iteratively using the above value as the first approximation. The results for the case $N = 4$ are shown in Figures 1–3. The parameters chosen are as follows:

- Hardware mean computation between failures $(= 1/\lambda) = 10^9$
- Checkpointing overhead $(cpo) = 1{,}000$ instructions per checkpoint
- $C = 3$, which makes $PR_1 = 50\%$ at $ipcf = 0$ and RB-size $= cps/2$
- $D = 30$ and $u = 3$, which makes $d = 1.0$ at $ipcf = 0.32$.

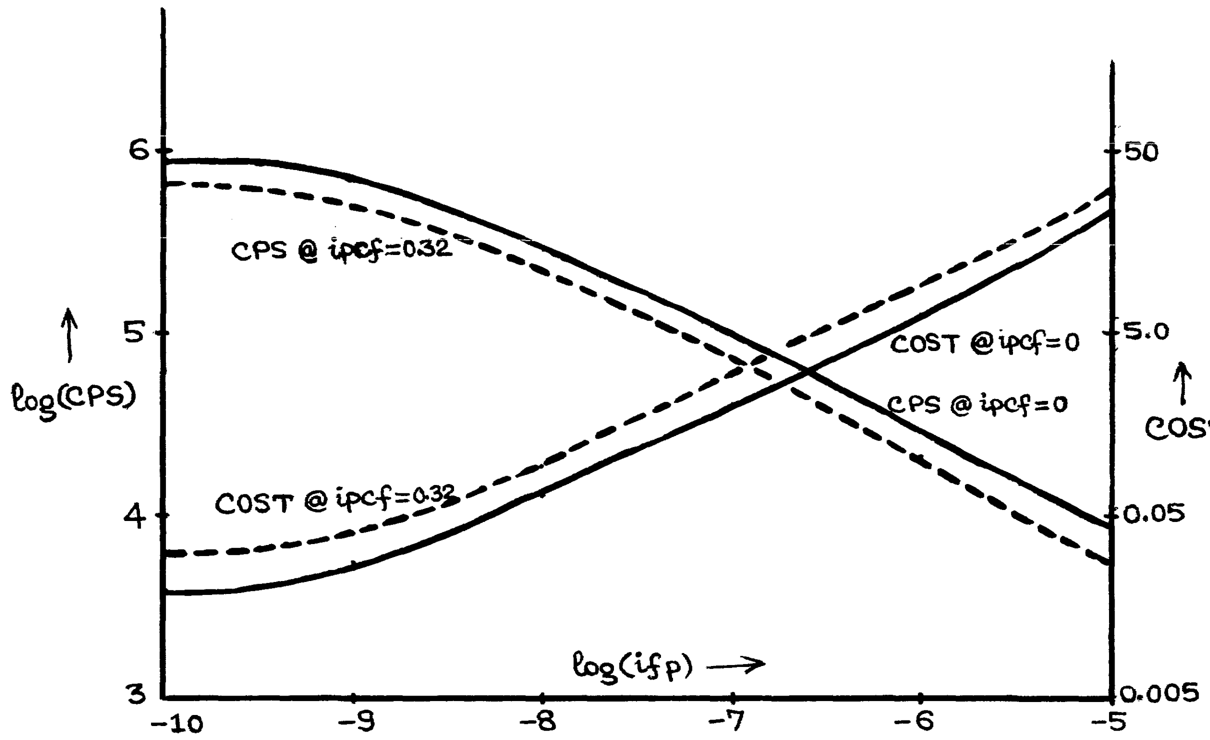


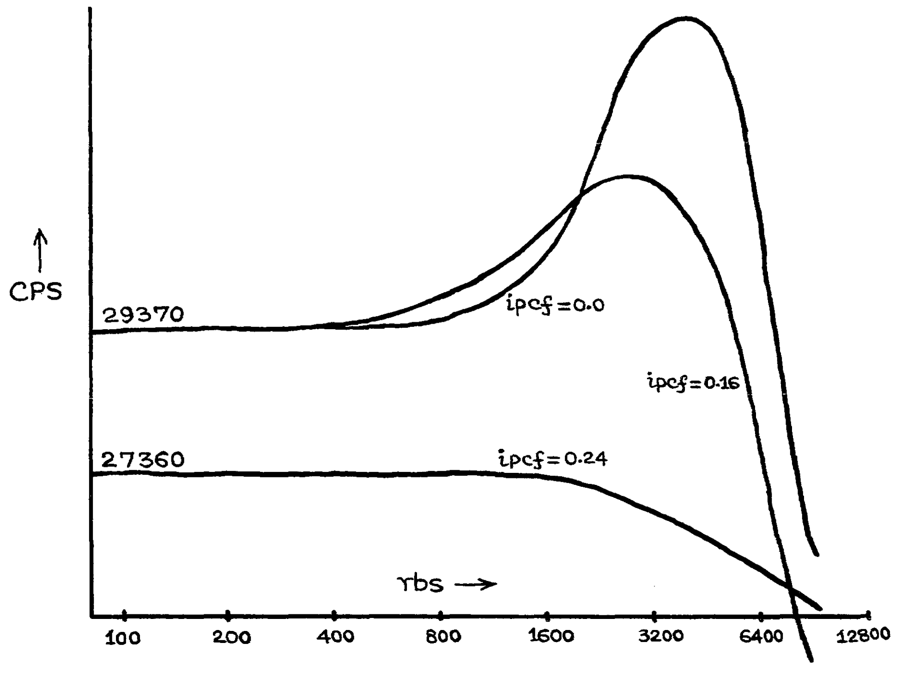
Figure 1—COST and CPS versus instruction failure probability

The figures show optimum *cps* (CPS) and optimum cost (COST) as a function of *ifp, rbs,* and *ipcf.* As expected, CPS decreases and COST increases with increasing *ifp.* It is interesting to note that for $ifp >> \lambda$, that is, when the effect of hardware failures is negligible, COST is directly proportional and CPS is inversely proportional to the square root of *ifp.* Since COST represents the fraction of instructions that are "useless," it must be $<< 1$ for a practical case. This means that for the above parameter values, the case $ifp > 10^{-6}$ would be highly undesirable.

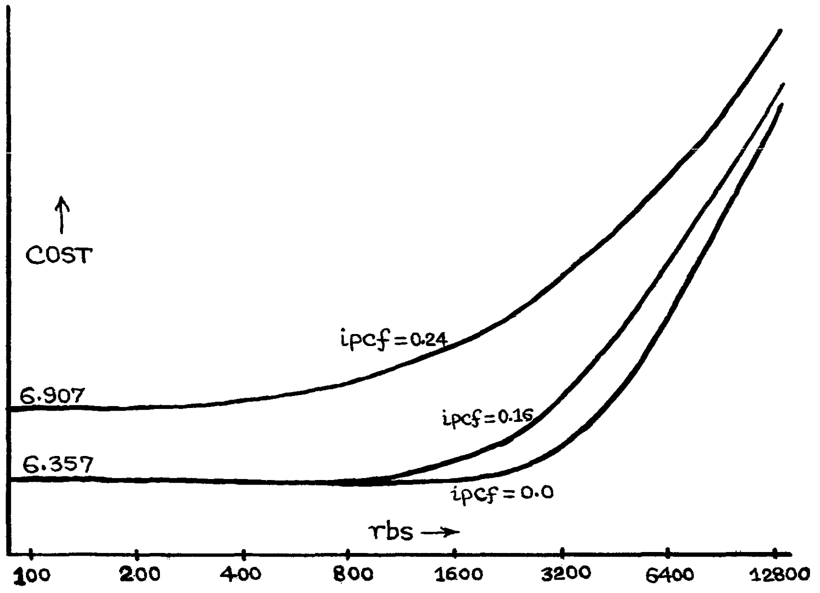
Both CPS and COST show interesting behavior with respect to c ($= C^*rbs$). First consider the case $ipcf = 0$. For $c << cps$, both COST and CPS are insensitive to increase in c but rise at an increasing rate for larger values. Although COST maintains its increasing trend, CPS peaks at around $c = cps/2$ and then falls rapidly. Such behavior is expected because as RB size increases correct recovery becomes increasingly costly. The reason for the peak in CPS is that the probability of correct execution (or reexecution) decreases as *cps* increases; therefore, even though a larger *cps* would make correct recovery more likely, the success of correct reexecution becomes more doubtful. Thus we cannot keep increasing *cps* indefinitely to cope with larger *rbs* values. As the interaction between processes increases, the recovery cost rises at an increasing rate and smaller *cps* is required in order to hold COST down. This also explains why the peak in CPS becomes smaller and shifts to left as *ipcf* increases. The results for other values of N (not included here) are very similar.

D. COMPARISON OF LOCAL AND GLOBAL CHECKPOINTING

As mentioned before, the motivation for introducing GCP is to reduce the cost and complexity of performing backward recovery in concurrent programs. When comparing GCP against LCP, we must consider two aspects: (1) the complexity and overhead of the mechanism itself, and (2) the cost of all computation that had to be discarded or redone. It is clear that the recovery mechanism used by the GCP scheme is very simple and would result in very little time and space overhead. On the other hand, the LCP scheme requires maintaining both a very large number of local checkpoints for each process and the complete history of the interprocess interaction during which they were established. Moverover, the determination of rollback points requires either a search through this long history (as in Kant and Silberschatz[4]) or an incremental update of information regarding it (as in Kim[5]). Thus the GCP wins in this respect. The argument tilts in favor of LCP when we consider the second aspect, because LCP determines the correct rollback points on the basis of process interaction history rather than by a trial and error method. Thus LCP would be expected to usually involve less recomputation than GCP. However, it should be noted that even in LCP we do not know exactly where the problem lies (if we did, it should have already been removed!) and the algorithm for determining rollback points will usually require significantly more rollback than necessary.



a



b

Figure 2—CPS versus recovery block size

It is clear from these observations that neither of the two approaches can be claimed to be always superior to the other. However, it can be argued that they are complementary if we consider the full range of interprocess interaction levels. Since all processes are rolled back to a common point in the GCP scheme, its overall cost would be higher than that for the LCP scheme for a system of processes that rarely interact. Furthermore, if the process interactions are rather infrequent, the overhead of maintaining and searching interaction history of the processes will be reasonably small. Thus LCP is an attractive scheme at low process-interaction levels. GCP appears to be a better scheme at moderate to high process-interaction levels, since the cost of maintaining and searching interaction history rises sharply with interaction level. A combination of the two schemes may also be used to advantage in certain situations. For example, we may carry out the determination

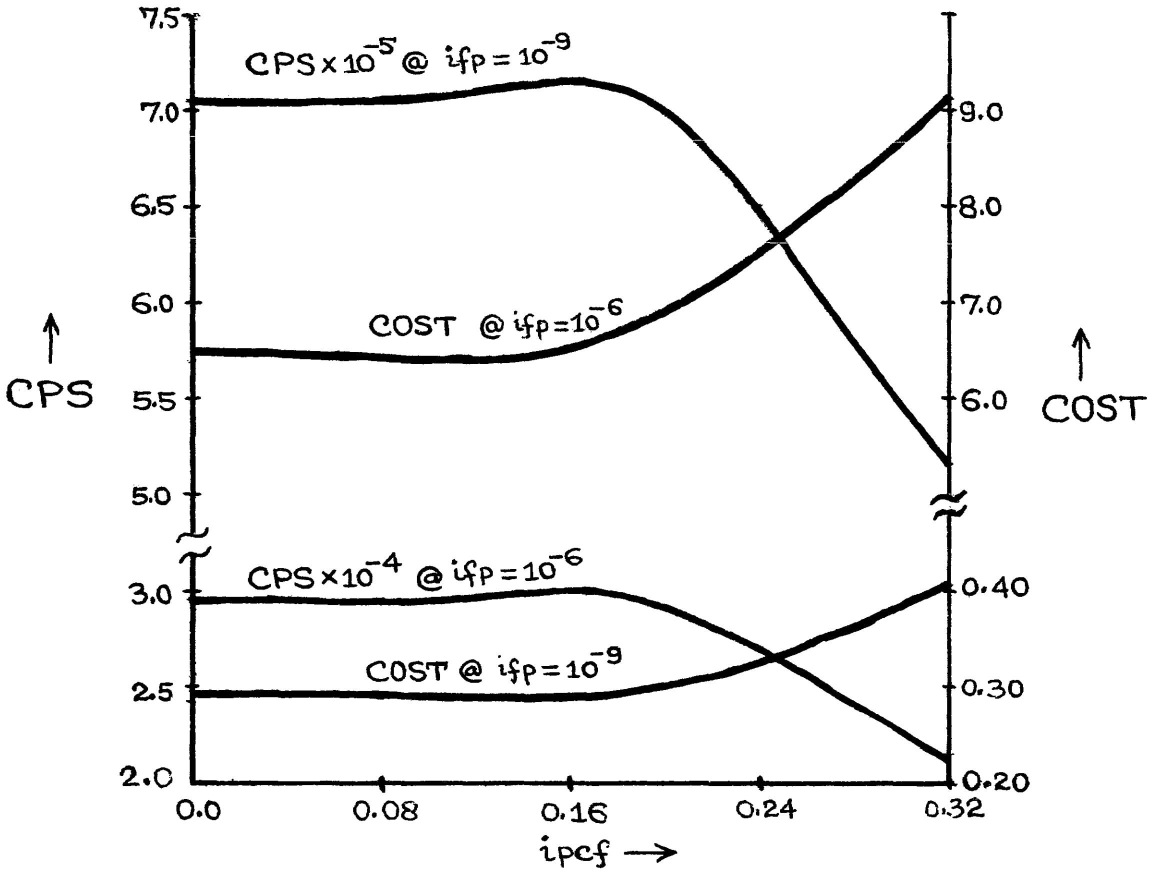


Figure 3—COST and CPS versus interprocess communication fraction

of rollback points for all processes assuming the LCP scheme but actually use GCP for rollbacks. In this case, the global checkpoint to which the system must be rolled back will be the one established prior to the earliest point in time to which some process must be rolled back according to our rollback-point determination algorithm.

E. CONCLUSIONS

In this paper, we have presented a global checkpointing scheme as an alternative to the local checkpointing inherent in the RB construct proposed by Randell. The basic purpose of introducing GCP is to simplify backward recovery in concurrent programs at the risk of discarding some computation unnecessarily. As noted above, this scheme is most suited for concurrent programs with moderate to high process-interaction levels. In this sense, the scheme complements the LCP scheme, which was designed primarily for sequential programs.

Several models for finding the optimum checkpoint interval have been proposed in the literature.[2,3] However, these models only consider recovery from transient failures where a rollback to the last checkpoint is sufficient for recovery. Our model deals with both transient hardware failures and the failures caused by software imperfections. (We do not consider permanent hardware failures, because the nature of recovery is very different in those cases). The model does take into account crucial system parameters such as recovery block size, the complexity and extent of process interaction, and so on. The software reliability was accounted for by a single parameter *ifp*, the instruction failure probability. An interesting extension would be to use some of the software quality metrics proposed in the literature to get a better characterization of software failure. Many other extensions are possible, such as the consideration of several classes of hardware and software faults, taking into account the effect of system load on failure rates, and so on.

REFERENCES

1. Beaudry, M. D. "Performance Related Reliability Measures for Computing Systems," *IEEE Transactions on Computers*, C-27 (1978), pp. 540–547.
2. Chandy, K. M. "A Survey of Analytic Models of Rollback and Recovery Strategies," *Computer*, 8 (1975) May, pp. 40–41.
3. Gelenbe, E. "On the Optimum Checkpoint Interval," *Journal of the ACM*, 26 (1979), pp. 259–270.
4. Kant, K. and A. Silberschatz. "Software Fault Tolerance in Concurrent Systems." Technical Report, Northwestern University, September 1982.
5. Kim, K. H. "An Approach to Programmer Transparent Coordination of Recovering Parallel Processes and its Efficient Implementation Rules," *Proceedings of the International Conference on Parallel Processing*, August 1978, pp. 58–68.

6. Musa, J. D. "The Measurement and Management of Software Reliability." *Proceedings of the IEEE,* 68 (1980), pp. 1131–1143.
7. Randell, B. "System Structure for Software Fault Tolerance." *IEEE Transactions on Software Engineering,* SE-1 (1976), pp. 220–232.
8. Schneider, V. "Some Experimental Estimators for Developmental and Delivered Errors in Software Development Projects." *Proceedings of COMPSAC 80,* Oct. 29–31, 1980.

Development tools for bus controller software

by M. I. THOMAS
TECSI-SOFTWARE
Paris, France

ABSTRACT

This paper addresses the problem of software specification and generation for bus controller software. This software is representative of a class of software modules for which the tools may be used. Two tools are described, a simple language in which to specify the module and a program generator that produces code directly from the specifications. The language uses finite state diagram ideas as do many other specification languages, but is constrained so that generation of high-quality code is feasible. A brief outline of the structure of the code generated is given, followed by some indications of the performance of the tools and the experience gained from their use.

INTRODUCTION

This paper describes two program-development tools that have been constructed for use in an avionics software project. The tools are a language for specifications of a data-bus system and a program generator used to convert the specifications into code. They were developed for their utility in the specific problem domain rather than with more general objectives. Nevertheless they illustrate, in practical running software, the advantages to be gained by formal system specification and its automatic conversion into code.

The data-bus standard of MIL-STD-1553B defines a configuration consisting of a bus controller that supervises data traffic on a bus to which a number of peripherals may be attached.[1] The controller uses internal data structures to decide which source and destination devices are appropriate for a data transfer request. In effect, these data structures model the current state of the peripherals attached to the bus. Certain signals, for example a notification of device failure, change the values in the data structures and will consequently affect the treatment of subsequent transfer requests.

The classical paradigm for problems of this nature is the state diagram. It is difficult to apply in this particular case because of the large number (up to 30) of peripherals, each of which may assume many different states. (An average of n states for each peripheral gives a state space of n^{30} configurations. Because the states of the peripherals are largely independent, the size of the state space is not greatly reduced by constraints on interrelations). Any reasonably complete state diagram or tabular representation is clearly impractical in these circumstances.

The consequent difficulties facing the project were those of

1. Specifying the controller software responsible for maintaining a correct model of the current bus state.
2. Specifying how bus signals were to be processed taking account of the information in the system model.
3. Verifying the coherence of these specifications.
4. Producing a program embodying these specifications in such a way that small changes in the specification (which may be frequent in a development environment) could be reliably and quickly incorporated into the program.

These difficulties were resolved to a greater or lesser extent by two complementary tools that have been developed. The first tool is a simple language that can be used to specify the bus controller software. It has facilities to describe the data structures used by the bus controller to model the system state. The language may also be used to describe how the controller must react to bus signals in the light of the current state of its model. The second tool is a program that accepts a specification written in the language and generates variable declarations and code to meet the specification.

The next section of this paper describes the language that has been developed and its relation to other specification languages and models. The third section outlines some features of the program-generator tool. This is followed by a brief description of the experience gained by the use of these tools.

THE SPECIFICATION LANGUAGE

Overview

The language was designed to allow the specification of software belonging to a relatively small subset of systems. It does not attempt to describe hardware components or the interactions and synchronizations between multiple software components, nor does it contain facilities to reflect timing constraints. It is clearly not intended to serve as a general system-specification language.

It does permit the description of an individual software module and the behavior required from it in response to the various inputs that it may receive. These inputs may come from one or several software modules. It is assumed that the processing of each input to the specified module is completed before processing of the next input starts. This means that the problems of deducing the access protection necessary for the module's common data structures are avoided, since concurrent accesses are precluded.

These restrictions of the language were adopted for the pragmatic reason that the resultant language would adequately specify the bus controller software under development, and to facilitate the generation of code directly from specifications written in the language. There are some more general specification languages that permit the generation of code skeletons for the specified system either automatically or by a straight-forward manual process,[2] but a considerable amount of information needs to be added to these code skeletons.

A Simple Example of Part of a System Specification

It is important to note that any example, such as the one given here, that is small enough to be understood easily can also be represented efficiently in a tabular structure. As the size of the example grows, the tabular structure becomes more unmanageable.

The example incompletely specifies the behavior required from a software module controlling part of the electrical system of a car. When the car's ignition switch is on, the direction indicator switch activates the flashing direction indicators. When the ignition switch is turned off and the direction indicator switch is in the left or right position, the car's electrical system illuminates the left or right parking light (unless the parking light switch is on, in which case both lights are illuminated).

```
COMPONENTS
IGNITION = (ON, OFF);
INDICATOR-SWITCH = (LEFT, NEUTRAL, RIGHT);
PARKING-SWITCH = (ON, OFF)

SIGNALS
ACTIVATE-INDICATOR-LEFT,
ACTIVATE-INDICATOR-RIGHT,
DEACTIVATE-INDICATOR,
IGNITION-SWITCHED-ON,
IGNITION-SWITCHED-OFF,
PARKING-SWITCHED-ON,
PARKING-SWITCHED-OFF

RULES
(IGNITION = ON) & ACTIVATE-INDICATOR-LEFT
⇒
  [START LEFT INDICATOR FLASHING]
   (INDICATOR-SWITCH = LEFT);

(IGNITION = OFF,
PARKING-SWITCH = OFF,
INDICATOR-SWITCH = LEFT) &
 DEACTIVATE-INDICATOR
⇒
  [TURN OFF LEFT PARKING LIGHT]
   (INDICATOR SWITCH = NEUTRAL);

(PARKING-SWITCH = OFF,
INDICATOR-SWITCH = LEFT) &
 IGNITION-SWITCHED-ON
⇒
  [TURN ON ALL IGNITION SYSTEMS]
  [TURN OFF LEFT PARKING LIGHT]
  [START LEFT INDICATOR FLASHING]
   (IGNITION = ON);
etc.
```

A specification contains three major subdivisions, namely the COMPONENTS, SIGNALS, and RULES sections.

The COMPONENTS section serves to define the data structures manipulated by the module being specified and used by the module to determine its responses to the input signal it receives. In the case of the bus controller these data structures may represent the state of peripherals attached to the data bus. It is also possible to define data structures that do not reflect the state of physical devices; for example a data structure that indicates the current flight phase could take the values TAKE-OFF, LANDING, CRUISE, and COMBAT.

FLIGHT-PHASE =
(TAKE-OFF, LANDING, CRUISE, COMBAT)

Two types of data structure, simple and complex, may be defined in the COMPONENTS section. Simple data structures are analogous to PASCAL's enumeration types. The data structures IGNITION, INDICATOR-SWITCH, and PARKING-SWITCH in the example are all simple. Complex data structures are a way of grouping a set of related simple and complex data structures in order to refer to them by name.

Where a module's data structures represent the states of entities outside the module they need only reflect those classes of states that can influence the actions of the module. That is, the internal states of the entity can be partitioned and each class of the partition be considered as one state from the point of view of the module. This approach is common in specification languages.

The SIGNALS section contains a list of input signals that may be received by the module. The input signals are not parameterized in any way, so an input signal whose parameters cause radically different actions to be performed should be characterized by different input-signal names in this section. In fact, the list may also contain signals which are generated internally within the module. These signals can be used to direct the sequencing of the processing of external inputs to the module.

The RULES section associates states of the module's data structures and an input signal with the processing that needs to be executed and the new state of the module's data structures. The left-hand side of each rule (preceding⇒) defines a subset of the state space of data structures, called the source set, and also contains the name of an input signal. The right-hand side of each rule contains an optional list of the names of actions to be taken and the new values to be assigned to the data structures, called the destination state. Only those data structures that actually change value need to be mentioned.

The semantics of a rule are clearly related to the state diagram paradigm:

> Whenever the signal is to be processed by the module and the data structure values belong to the source set than the rule is applicable. The named actions should be carried out and then the data structures should assume the new values given.

Each rule therefore defines a class of transitions on the state diagram whose states reflect all the possible configurations of the data structures.

As indicated in the example, the source set need not indicate values for all of the module's data structures. It is also possible to specify that a data-structure value should be one of a subset of its possible values or that it should not take a certain value or a subset of values. For example,

(FLIGHT-PHASE < > COMBAT)
(FLIGHT-PHASE = (TAKE-OFF OR LANDING)).

Neither of these forms is permitted in the description of the destination state since it makes no sense to assign any one of a set of values to a data structure.

There are situations in which the actions in a rule may change the new value to be assumed by the module's data structures. For example, in a printing module there may be a data structure

END-OF-PAGE = (TRUE, FALSE).

The rule describing the printing of a line may be

```
(END-OF-PAGE = FALSE) & LINE-TO-PRINT
⇒
  [PRINT LINE]
  [DECREMENT NUMBER OF LINES REMAINING]
   (END-OF-PAGE = FALSE);
```

The action of decrementing the number of lines remaining may yield the result zero, in which case END-OF-PAGE should become TRUE. It is possible to indicate that the value to be assigned to a data structure may be changed by an action so that the code-generation algorithm may take appropriate action.

The language also provides facilities for the definition of data structure types in a manner analogous to type definition in PASCAL or ADA. These definitions appear in an optional TYPES section.

Comparisons with Other Specification Languages

The term specification language as it is currently used covers two broad classes of language. The first is requirement-specification languages.[3,4,5] The system to be specified is described from an external point of view, though reference may be made to subsystems in order to clarify the description. The second class is system-specification languages, where the system whose design is to be specified is decomposed into subsystems.[6] Each of these subsystems is defined together with its relationship to other subsystems. This paper describes a specification language for use at a later stage in the development process, where the behavior of individual software components that are regarded as non-decomposable must be specified. Formalization of the interaction between such components, though of primary importance in system-specification languages, is not addressed here. The description of system components using the concept of internal state is common in specification languages.[2,5,7] However, the decomposition of these states into independent data structures whose combinations define the state space of the component is not present in these systems.

The notion of a system stimulus is also used in specification languages.[3,7] The interpretation in the language specified here is identical to its usage in these other languages.

An older technique for specification of software modules is the decision table.[8] Although decision-table conditions can be regarded as specifying the subset of states in which some actions are to be performed, the idea of a new state to be entered after the actions is not present.

Verification of the Specification

One of the advantages of a rigorous formal specification is that verification techniques may be applicable, though the current system does not contain a verification tool. The only check that is currently carried out ensures that at most one transition rule is applicable from any system state in response to any input signal. In manually written programs this situation is often disguised by dependencies on the order in which the various tests of the data structures are made. This section will indicate some verification techniques that are readily applicable to specifications formulated in the language.

It must be assumed that the data structures of a module are independent, that is, that manipulation of one does not necessarily change the value of others. A bus-controller module where the data structures model independent peripherals attached to the bus is an example of such a case. Such systems can be concisely modeled using Petri Nets.[9] Several existing languages for system description use Petri Nets for validation purposes.[10]

Each state of each data structure may be represented by a place in a Place/Transition Net. The transitions of the net represent the transition rules of the specification. The marking graph of these places defines the state space of possible configurations of the data structures. Clearly the Place/Transition net so far described is an alternative representation of the state diagram of the system and needs to be completed with a description of the signals and the order in which they may arrive for processing by the module. The new representation does however permit the application of analyses of structure that would be more difficult with state diagram representations, while the fact that the net represents a state diagram reduces the complexity of these analyses.

The use of such verification techniques implies an extension of the language to cover descriptions of possible input-signal orderings. Riddle has developed a formalism to describe such orderings.[11] More advanced verification may include the division of the state space of data-structure configurations into classes of a spectrum ranging from Impossible through Acceptable Malfunction to Complete Malfunction using the concepts of deontic logic.[12]

THE PROGRAM GENERATOR

This software tool was developed to convert specifications into high-quality code. It also generates declarations for the data structures defined in the specifications. PASCAL or CORAL66 code may be generated. Since the generator uses an intermediate representation of the program, other target languages can be easily added.

The generated program has a structure analogous to PASCAL's *case* statement, where the case branch selection is made on the input signal that is to be processed. This does not represent a loss of generality, since the input signals can be represented as the values of a data structure in the COMPONENTS section and the SIGNALS section can be redefined to contain only a single input signal that is the notification that an input event has occurred.

When the transition rules of a specification have been partitioned into classes according to the input signals under which they may apply, code is generated for each class. The order in which the tests of data structure values are made greatly affects the efficiency of the generated code. An exhaustive search of possible orderings even within a subset of rules is not possible for combinatorial reasons, so heuristic methods are used.

A discussion of the problems of code generation and the algorithms used can be found in another of the author's articles.[13]

EXPERIENCE WITH THE TOOLS

The avionics software project for which the tools were developed is divided into phases. At the end of the first phase a hand-coded version of a simple bus controller had been prepared. This hand-coded version used a tabular, packed data representation of the states and transitions of the bus controller. The tools were also completed towards the end of the first phase. In order to gain experience, the simple controller was specified in the defined specification language. This took approximately two weeks. However, no comparison with the time taken to specify the hand-coded version is possible since no formal specification of it existed.

The program and data of the tool-generated version occupied approximately the same amount of memory as the original, but the execution time for a sequence of test inputs was reduced by approximately 30%. It is difficult to evaluate this result, since the automatically generated version does not use packed data.

The time taken for the tool to process a set of rules obviously depends on their characteristics, but sets of one hundred rules are typically processed in approximately three minutes of CPU time on a DEC Vax computer.

REFERENCES

1. MIL-STD-1553B. U.S. Department of Defense, Sept. 1978.
2. Ludewig, J. "Computer Aided Specification of Process Control System." *Computer,* 15 (1982), 5, pp. 12–20.
3. Taylor, B. J. "A Method for Expressing the Functional Requirements of Real Time Systems." *Proceedings of the IFAC/IFIP Workshop on Real-Time Programming.* Leibnitz, Austria, April 1980, pp. 111–120.
4. Teichreow, D. and E. A. Hershey. "PSL /PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems." *IEEE Transactions on Software Engineering,* SE-3 (1977), 1, pp. 41–48.
5. Davis, A. M. "The Design of a Family of Application-Oriented Requirements Languages." *Computer,* 15 (1982), 5, pp. 21–28.
6. Riddle, W. E. "An Assessment of DREAM." In H. Hunke (ed), *Software Engineering Environments.* Amsterdam: North-Holland, 1981.
7. Alford, M. W. "A Requirements Engineering Methodology for Real-Time Processing Requirements." *IEEE Transactions on Software Engineering,* SE-3 (1977), 1, pp. 60–69.
8. Metzner, J. R. and B. H. Barnes. *Decision Table Languages and Systems.* New York: Academic Press, 1977.
9. Peterson, J. L. *Petri Net Theory and the Modeling of Systems.* New York: Prentice-Hall, 1981.
10. Jorrand, P., J. P. Queille, and J. Sifakis. "Conception et Vérification des Applications Réparties: Présentation du Système CESAR et de développements en cours." *Actes des journées BILAN ET PERSPECTIVES DES 20-21-22 JANVIER 1982—Projet Pilote SURF sur la sûreté de fonctionnement des systèmes.* Paris: Agence de l'Informatique, 1982.
11. Riddle, W. E. "An Approach to Software System Behaviour Description." *Computer Languages,* 4, (1979), 1, pp. 29–47.
12. Anderson, A. R. "The Formal Analysis of Normative Systems." In N. Rescher (ed.), *The Logic of Action and Decision.* Pittsburgh: University of Pittsburgh Press, 1967.
13. Thomas, M. I. "Automatic Generation of Bus Controller Software from its Specification." Unpublished manuscript, 1982. (Submitted for publication.)

Logic analysis and its tools

by Dr. R. S. WANG
RCA
Moorestown, New Jersey

ABSTRACT

This article discusses the logic analysis of a program, points out problems with the logic analysis process in general use, and introduces an approach to logic analysis that is more effective and less time consuming. The new method generates logic paths out of programs and preanalyzes the paths in lieu of directly analyzing the program. Three preanalysis software tools are introduced: procedure logic path generator, program logic path generator, and logic path preanalyzer. Sample outputs are given to demonstrate the difference between program analysis and logic path analysis.

INTRODUCTION

The general process of analyzing a program consists of identifying entry point, identifying the path at branch point, going to the called routine, returning to the calling routine, manipulating data, and interpreting data. The process is completely undisciplined[1] and is laborious and repetitive most of the time.

The logic analysis is done by different people at different stages of program development. The typical and more thorough logic analysis is done during the design stage by the designer and reviewer, and during the test stage by the tester. The efficiency and effectiveness of the logic analysis approach will cause a distinct impact on the quality and cost of the design and test.

This article suggests a systematic logic analysis approach. The approach is to isolate and preanalyze the program logic paths on the computer before further analysis. The logic analysis of a CMS-2 language program in the white box test stage is used for the illustration.[1]

LOGIC ANALYSIS FOR MODULE VERIFICATION TEST

A system consists of a set of modules with each module consisting of a set of procedures. The test of a module is a white box test by which data presets and data outputs are needed to be defined through the logic analysis of the module and the interpretation of the test case in the test requirement. Two approaches are used to complete the module verification test (MVT): One is a machine test and the other is an inspection test.

For each test case, after the tester has acquired the understanding of the overall function to be performed, the tester begins to analyze the program and the related data design. The tester tries to correlate the test case and the program to make sure that the program can carry out the functions as described in the test case. The analysis consists of the following processes:

1. Defining data according to the preset data specified in the test case and the format required by the data design
2. Justifying and defining the preset data needed by the program yet not defined in the test case
3. Manipulating the data as instructed by the related statements in the program
4. Choosing branches at decision statements
5. Comparing the outputs of the program with the expected outputs specified in the test case

The above analysis process may iterate one or more times because of the following reasons:

1. Mistakes made in the data manipulation
2. Imperfection of the test case definition
3. Complexity of the program
4. Confirmation of the analysis result
5. Identification or confirmation of the discrepancies or errors

Problems of the Logic Analysis of MVT

The analysis is the foundation of the MVT. It is usually quite time-consuming for the tester, requiring much patience to go through the instructions primarily executed by the CPU. Figure 1 symbolically represents the nature and problems of the program analysis. Part (a) of Figure 1 represents a set of test cases. Input and output data are specified in the test case. The stars symbolize the special data format used in the test case. Part (b) represents the program where many logic paths are blended together in a box. The data, which are the inputs and outputs of the logic paths, are expressed in definite formats as symbolized by the circles. Each test case in Part (a) will have a corresponding logic path in part (b).

A problem of the prevalent logic analysis is that, for each analysis, one has to identify the tangled logic path of the program while performing other analysis efforts. This impacts the required time and quality of the total analysis.

For short and straightforward logic paths, the problem is not severe. However, for lengthy and tangled logic paths, the problem is severe. For each analysis one has to memorize the test conditions, select branch at decision statements, manipu-

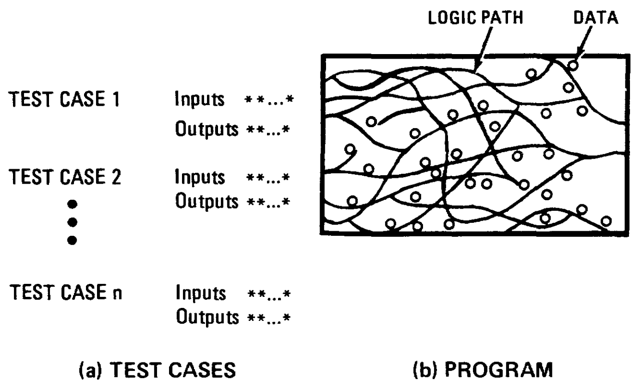


Figure 1—Symbolic representation of test cases and program

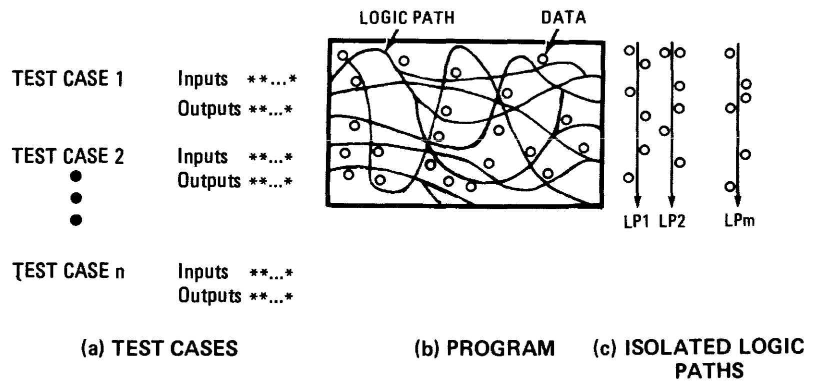


Figure 2—Symbolic representation of test cases, program, and isolated logic paths

late data, flip the pages to locate the called procedures and calling procedures, and memorize control indicators. Because of the involvements, one may easily lose track and end up repeating the analysis or obtain analysis of questionable accuracy. The uncertainty may appear in the result of machine test. It may result in another cycle of analysis and machine test. For inspection test, this simply means a questionable test quality. One of the causes of the problem is directly analyzing the program each time.

In doing the analysis, visibility is a critical factor that affects the analysis effectiveness.[2] Because the program contains many logic paths not related to the test case, the related logic path scatters around in the program, and the visibility is greatly impacted.

LOGIC PATH ISOLATION

To increase the visibility, to save time, and to improve the analysis effectiveness, it is helpful if logic paths can be isolated from the program for further analysis. Logic path isolation is symbolically depicted in Figure 2. The program shown in Figure 2(b) is transformed into a set of isolated logic paths shown in Figure 2(c). For further analysis, one can check the test case against the isolated logic path instead of checking the test case against the whole program.

The decision statements in the path become condition descriptors. They reflect the preset conditions, data, and the derivatives of the preset conditions and data. After the logic path is isolated, to analyze a logic path one does not have to flip the pages back and forth to locate a called procedure or a calling procedure. One does not have to select a branch at every decision statement either. The analysis of the logic path is simplified and straightforward. The tester can make a direct comparison between the conditions indicated in the decision statements with the data presets in the test case. The expected outputs of the test case can be compared directly with the outputs of the imperative statements. The redundancy of the logic path identification effort is eliminated. The visibility of the logic is greatly increased. Software tools to carry out the logic path isolation processes are discussed in the following paragraphs.

Procedure Logic Path Generator

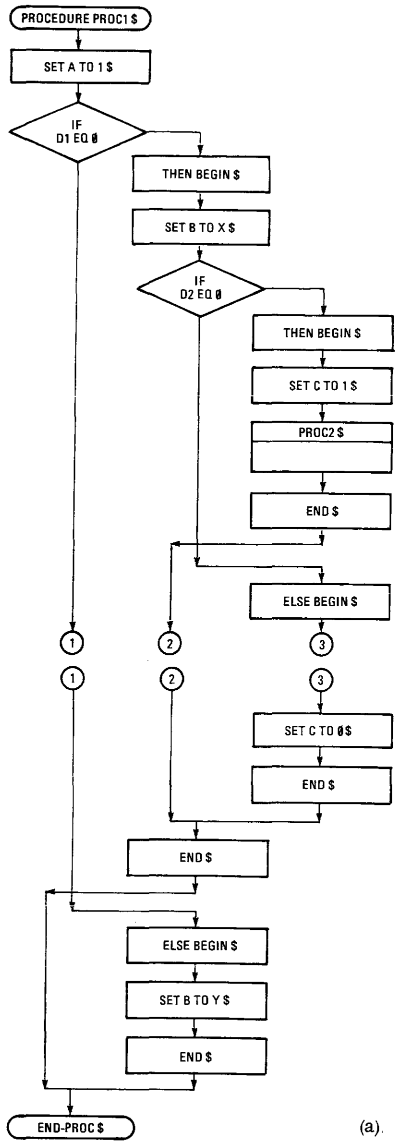
This generator uses the program as input. It converts each procedure into a set of logic paths. Each logic path is identified. Files are created. Printed output is also available. An example is shown in Figure 3 to illustrate the process. A software system has one module which consists of two procedures, PROC1 and PROC2, as shown in Figure 3(a) and 3(b). By using the program as input, the generator produces one set of three logic paths for PROC1 as shown in Figure 3(c), and one set of three logic paths for PROC2 as shown in Figure 3(d). The decision statements are marked so that they can be analyzed separately and do not interfere with the imperative statements. The decision statements in the program become condition descriptors in the generated logic path. The END statements of the program are not shown in the logic path. Since all the unrelated information is not shown in the generated logic path, obviously the visibility is significantly increased. For each logic path, analyzing the generated logic path is much easier and more effective than analyzing the logic path in the program procedure.

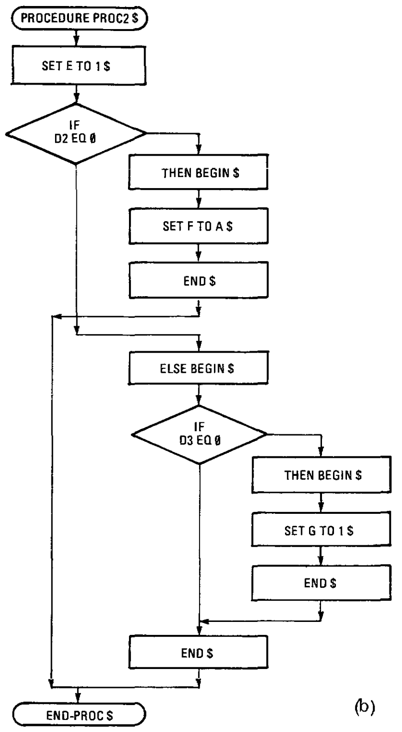
Module Logic Path Generator

After the logic path has been identified for a test case a sequence of procedure logic path IDs are manually listed. This sequence of procedure logic path IDs is entered as the input to the module logic path generator. The generator then accesses the procedure logic path files for the selected IDs. The procedure calls can be resolved automatically during the generation process. The module logic paths are shown in Figure 4. Like the procedure logic paths, the module logic paths present a higher visibility for logic analysis than the logic paths in the program.

Logic Path Preanalyzer

The outputs of the procedure logic path generator and the outputs of the module logic path generator can be further organized by a tool called a logic path preanalyzer. The pur-





Logic Path 1

SET A TO 1
*IF D1 EQ 0THEN BEGIN
SET B TO X
*IF D2 EQ 0 THEN BEGIN
SET C TO 1
PROC2

Logic Path 2

SET A TO 1
*IF D1 EQ 0 THEN BEGIN
SET B TO X
*IF D2 EQ 0 ELSE BEGIN
SET C TO 0

Logic Path 3

SET A TO 1
*IF D1 EQ 0 ELSE BEGIN
SET B TO Y

(c)

Logic Path 1

SET E TO 1
*IF D2 EQ 0 THEN BEGIN
SET F TO A

Logic Path 2

SET E TO 1
*IF D2 EQ 0 ELSE BEGIN
*IF D3 EQ 0 THEN BEGIN
SET G TO 1

Logic Path 3

SET E TO 1
*IF D2 EQ 0 ELSE BEGIN
*IF D3 EQ 0 ELSE BEGIN

(d)

Figure 3—(a) Flowchart of PROCEDURE PROC1; (b) flowchart of PROCEDURE PROC2; (c) procedure logic paths of PROC1; (d) procedure logic paths of PROC2

Logic Path 1	Logic Path 2	Logic Path 3
SET A TO 1	SET A TO 1	SET A TO 1
*IF D1 EQ Ø THEN BEGIN	*IF D1 EQ Ø THEN BEGIN	*IF D1 EQ ØTHEN BEGIN
SET B TO X	SET B TO X	SET B TO X
*IF D2 EQ Ø THEN BEGIN	*IF D2 EQ Ø THEN BEGIN	*IF D2 EQ ØTHEN BEGIN
SET C TO 1	SET C TO 1	SET C TO 1
SET E TO 1	SET E TO 1	SET E TO 1
*IF D2 EQ Ø THEN BEGIN	*IF D2 EQ ØELSE BEGIN	*IF D2 DQ ØELSE BEGIN
SET F TO A	*IF D3 EQ ØTHEN BEGIN	*IF D3 EQ ØELSE BEGIN
	SET G TO 1	

Logic Path 4	Logic Path 5
SET A TO 1	SET A TO 1
*IF D1 EQ ØTHEN BEGIN	*IF D1 EQ ØELSE BEGIN
SET B TO X	SET B TO Y
*IF D2 EQ ØELSE BEGIN	
SET C TO Ø	

Figure 4—Module logic paths

Logic Path 1	Logic Path 2	Logic Path 3
Conditions	Conditions	Conditions
IF D1 EQ Ø THEN BEGIN	IF D1 EQ Ø THEN BEGIN	IF D1 EQ ØELSE BEGIN
IF D2 EQ Ø THEN BEGIN	IF D2 EQ Ø ELSE BEGIN	Inputs
Inputs	Inputs	Y, D1
X, D1, D2	X, D1, D2	Process
Process	Process	SET A TO 1
SET A TO 1	SET A TO 1	SET B TO Y
SET B TO X	SET B TO X	Outputs
SET C TO 1	SET C TO Ø	A, B
PROC2	Outputs	
Outputs	A, B, C	
A, B, C		

(a)

Logic Path 1	Logic Path 2	Logic Path 3
Conditions	Conditions	Conditions
IF D2 EQ ØTHEN BEGIN	IF D2 EQ ØELSE BEGIN	IF D2 EQ Ø ELSE BEGIN
Inputs	IF D3 EQ ØTHEN BEGIN	IF D3 EQ ØELSE BEGIN
D2, A	Inputs	Inputs
Process	D2, D3	D2, D3
SET E TO 1	Process	Process
SET F TO A	SET E TO 1	SET E TO 1
Outputs	SET G TO 1	Outputs
E, F	Outputs	E
	E, G	

(b)

Figure 5—(a) The preanalyzed procedure logic paths of PROC1; (b) the preanalyzed procedure logic paths of PROC2

pose is to make the analysis of generated logic paths easier. Examples are shown in Figures 5(a), 5(b) and 6. In Figures 5(a) and 5(b) the preanalyzed procedure logic paths are shown. The conditions and process in the logic path are listed separately. The inputs and outputs of the logic path are also identified. The input and output data may greatly facilitate the procedure interface analysis.

In Figure 6 the preanalyzed module logic paths are shown.

Logic Path 1	Logic Path 2	Logic Path 3
Conditions	Conditions	Because of the reason for Logic Path 2, this logic path is also considered invalid.
IF D1 EQ ØTHEN BEGIN	IF D1 EQ Ø THEN BEGIN	
IF D2 EQ ØTHEN BEGIN	IF D2 EQ Ø THEN BEGIN	
Inputs	IF D2 EQ Ø ELSE BEGIN	
X, D1, D2	IF D3 EQ Ø THEN BEGIN	
Process	It is apparent that the second and third conditions are contradictory. Therefore, this logic path is invalid.	
SET A TO 1		
SET B TO X		
SET C TO 1		
SET E TO 1		
SET F TO A		
Outputs		
A, B, C, E, F		

Logic Path 4	Logic Path 5
Conditions	Conditions
IF D1 EQ ØTHEN BEGIN	If D1 EQ Ø ELSE BEGIN
IF D2 EQ Ø ELSE BEGIN	Inputs
Inputs	Y, D1
X, D1, D2	Process
Process	SET A TO 1
SET A TO 1	SET B TO Y
SET B TO X	Outputs
SET C TO Ø	A, B
Outputs	
A, B, C	

Figure 6—The preanalyzed module logic paths

The analysis of the logic path is straightforward. Using logic path 1 as an example, it shows that if the conditions $D1$ and $D2$ are 0, the A, C, E, and F will be set to 1. B will be set to X. X, $D1$, and $D2$ are the inputs needed by the logic path. The input and output data provide critical information for module interface analysis if more than one module exists in a system.

For logic path 2 the tester can easily tell that the second and third conditions are contradictory, thus, the logic path is considered invalid. The visibility of the generated logic paths is clearly much better than that of the program. Redundant page flipping and branch selection are eliminated from the analysis effort.

CONCLUSION

The approach introduced is used to isolate a logic path in a program for an MVT test case. The immediate purpose is to increase the visibility of the related program logic to be analyzed. Judging by the author's experience, the approach may improve the test quality and reduce the test cost. Moreover, the logic paths generated may constitute part of the MVT test results. They can be used for reviews and the analysis of integration test. The integration test is the next level of test after MVT.

Since the basic concept of this approach is to help analyze programs effectively and economically, the applicability of this approach is not limited in MVT. It can be applied to the procedure test and the integration test. It is even applicable in the design state and the maintenance stage. Since the logic analysis is the major effort in software development and maintenance, the approach deserves further discussion.

The design, design review, procedure test, module verification test, and integration test are consecutive processes in the program development cycle. Similar logic path analyses are done in each of the processes as was done for the module verification test. In fact, they all present more or less the same problems as mentioned above. The logic path isolation and analysis efforts that may be useful to the next process are not saved. Consequently one may have to repeat the work. This is a waste. For example, the designers must check the logic paths in the flowchart to compare with the description in the specification. The design reviewers also have to do the same. If the generated logic path analysis approach is used, the designer may generate the logic paths not only for his own use, but it will also be helpful to design reviewers who analyze the design. Furthermore, logic paths generated may be more suitable for presentation during the review meeting. Likewise, in addition to the usage in that test process, the procedure logic paths generated in the procedure test process may be used directly in the module verification test process.

For the MVT process, the module logic paths generated are extremely useful for the modular interface analysis of the integration test. Because of their higher readability, the machine outputs of the logic path generators and preanalyzer provide a good medium for reviews, presentations, and reports. As the validation process is pushed from the test stage into the design stage, the need for a good medium in the design stage becomes apparent.

In the test stage and the maintenance stage the program is analyzed and the tools are applicable. In order to apply the approach and develop tools for the design usage, it is recommended that a program design language be used in the design process.

In industry, long years of software development experience have demonstrated that people are not satisfied with an ad hoc development approach even though the ad hoc approach may allow more freedom and demand fewer tools. The additional freedom may allow people to see some products earlier. Fewer tools may give a feeling of saving some development cost. Now the state of the art is to impose more control and use more software tools. Generally, for large program development, better control enables analysts to see the final product earlier and the software tools can lower the total development cost.

REFERENCES

1. Myers, G.J. *The Art of Software Testing.* New York: John Wiley & Sons, 1979.
2. Van Tassel, D. *Program Style Design, Efficiency, Debugging, and Testing.* Englewood Cliffs, N.J.: Prentice-Hall, 1978.

MANAGEMENT

Bennet P. Lientz
University of California,
Los Angeles
Los Angeles, California

This track addresses specific concerns in information systems staffing, management, and planning. The subtrack on staffing includes sessions on the role of women in systems, software project management, and improving staff effectiveness and productivity. The second subtrack, on software maintenance, has sessions dealing with the management of software maintenance, applications of software engineering, technical issues in maintenance, and motivation of the software maintenance programmer. The final subtrack is on planning and control; it comprises a session on planning and one on the audit of complex computer/communication systems. All told, more than 50 speakers, with experience as developers, implementers, users, and managers of computer technology, will present papers or serve on panels in this track.

EDUCATION

Ronald S. Lemos
California State University
Dominguez Hills, California

In a relatively short time the computer has had a profound impact on educational processes throughout our society. Computer literacy is becoming necessary for effective functioning in an increasingly complex environment. This is especially true for people working in organizations. Computer literacy is discussed in terms of differences between what industry expects of a computer-literate employee and what higher education plans to produce. Another session deals with creative uses of educational technology. State-of-the-art uses of videodiscs in industry, schools, and universities will be discussed.

Improving software maintenance attitudes

by PAUL C. TINNIRELLO
The A. M. Best Company
Oldwick, New Jersey

ABSTRACT

Attitudes towards maintenance have been an overlooked source of problems in the software maintenance process. In the past, there has been little recognition of the significance of how attitudes affect the performance of maintenance functions. Investigation into the origin of these attitudes has led the author to formulate feasible solutions that foster productive attitudes through the educational and professional work environments.

INTRODUCTION

Progress in the development of software maintenance techniques has been languid in comparison to the growth in software development procedures. While fourth generation software promises to ease the maintenance difficulties, it does not change the fact that maintenance today is performed on software that has been developed in the past 25 years. It is not surprising, then, to discover that as much as 80% of all software costs are spent in maintenance effort while only 20% of the cost is invested in developing systems that will possibly have software simplicity.[1] The neglect in software maintenance development has placed a stigma on the maintenance process. In addition, there has been a serious impact on the performance of maintenance in the programming environment as a result of the attitudes arising from poor maintenance procedures.

DEFINING THE MAINTENANCE PROCESS AND IDENTIFYING THE PROBLEM AREAS

The software maintenance process can be interpreted as the correction, adaptation, and enhancement of computer programs and systems.[2] This definition of the maintenance process is widely accepted among those in the data processing (DP) community. However, finding agreement on what constitutes maintenance problems has been a stumbling block for years. Part of the difficulty in defining the problems stems from the way software maintenance is viewed.[3] Management may have a different concept of maintenance functions than a programmer who is directly involved with maintenance activities. Still another viewpoint may come from the end-user who has extracted a notion of maintenance from both management and programmers.

It is the opinion of the author that software maintenance problems can be segmented into three areas:

- maintenance management,
- maintenance programming,
- maintenance attitudes.

Maintenance management can be defined as the management of the software maintenance process within the computer-based organization. Maintenance management affects programmers, managers, and end-users, and requires the careful integration of all parties towards a successful solution, whether it be correction, adaptation, or enhancement. Maintenance programming can be defined as the technical methodology in which a correction, adaptation, or enhancement occurs. Such methodologies include programming practices and techniques implemented within existing software systems. Finally, maintenance attitudes can be defined as the position an individual has towards the software maintenance process in its entirety. Maintenance attitudes are usually held by many members of the computer-based organization, with the strongest attitudes being held by those who have the greatest interaction with the maintenance process.

INTERDEPENDENCIES IN THE MAINTENANCE SOLUTION

If there is to be any development in the software maintenance process, then each of the problem areas of maintenance management, maintenance programming, and maintenance attitudes must be improved. Recognizing this fact is, of course, much easier than agreeing on which area has the most impact on the maintenance process. Thus far, it appears as if the emphasis has been placed on the maintenance programming category. One of the approaches used in this area has been the evolution of structured programming techniques, which promised program maintainability through a modifiable and adaptable design. Academic institutions, especially DP organizations, began to stress the use of structured programming techniques with the naive hope that maintenance complexities would eventually by eliminated. Unfortunately, structured concepts have not eradicated all of the maintenance problems. They have eased, however, some of the complexities in the maintenance process.[4] In addition, attitudes in performing maintenance functions have improved for those individuals who are responsible for maintaining structured programs and systems. This improvement in attitude, which was elevated by the improvement in programming technique, demonstrates how each maintenance area is dependent on the other for success. The converse is also true. Poor maintenance management would affect the quality of work being performed and also diminish the attitude toward the maintenance process.[5]

At this point, the author would like to suggest that maintenance attitude is a problem segment that, when improved, can have more benefits for the maintenance process than improvements in the other problem areas. Until now, maintenance attitudes have been recognized only as a result of changes in the maintenance-management and maintenance-programming segments.

COMPLEXITY OF MAINTENANCE ATTITUDES

The dynamics of attitudes is not fully understood. Attitudes can be a mixture of emotional and mental processes that an individual develops through personal experience.[6] Attitudes towards software maintenance can affect maintenance management and maintenance programming in such a way as to possibly prevent either initiative or completion in one or both areas. The fact that software maintenance will not disappear, nor will there be an all-purpose cure, indicates an attitude unto itself. This complexity of attitudes makes it difficult to find an approach that would succeed in improving all attitudes towards the software maintenance process. No matter what approach is used, attitudes will always be derived from the experience that the individual encounters. It is with this thought that the author is only suggesting a feasible solution to the maintenance attitude problem. Further still, the author wishes to confine his suggestions to improving those attitudes of the computer programmer only.

ATTITUDE DEVELOPMENT

In an attempt to find a method for improving maintenance attitudes, it is necessary to uncover the origin of such attitudes. First, there is a need to examine those attitudes that grow out of the work experience. The majority of programmers usually encounter maintenance duties during the first several years of their professional careers. In some cases the maintenance work may be moderate to light, while in others the maintenance responsibilities can be heavy. Some organizations have a definitive policy with respect to new programmers that requires that they be assigned maintenance responsibilities in order to better their understanding of existing systems and to improve their software skills. This philosophy has been noted for its advantages by advocates of the software maintenance process.[3,5] However, such exposure to maintenance, especially with new programmers, can be detrimental to the organization and can possibly cause the development of poor attitudes about the maintenance process.[3] The point can be argued either way about maintenance benefits, but the net result in attitude is usually negative, even though some valuable experience was gained.

Perhaps this suggests that it is unpopular to have a good attitude toward maintenance work. In any case, the effects of maintenance have been recognized by programming managers involved with maintenance-related activities. Such effects include high turnover, low productivity, and excessive software costs.[7] Programmers who reflect the effects of the maintenance process often possess certain attitudes about their work. These high-level attitudes, as they will be termed, include boredom, defeatism, frustration, and a feeling of lack of recognition.[8,9] They are usually attributed to some set of conditions that is met while the programmers perform maintenance functions. These conditions or high-level factors, as they will be termed, include poor, little or no documentation; unstructured or poorly designed programs and systems; poor programming practices such as excess switches, meaningless data names, and nonstandard language commands; and extra work hours, odd work hours and pressure to complete maintenance tasks in little or no time. There is no doubt that the outgrowth of these high-level factors results in a poor or negative maintenance attitude. More important is the possibility that high-level factors, which foster high-level attitudes, also perpetuate the high-level factors.

As an example, consider a programmer who is given the task of performing a maintenance function on a poorly designed, poorly documented old program that has been the maintenance responsibility of twenty-five prior programmers. The probable result of such a task for the average programmer is that an attitude is either developed or supported against the maintenance process. In turn, the programmer will probably not provide any more insight into the program than was originally given. The author's experience has shown that the programmer's attitude will allow only minimal documentation and programming techniques to be performed, perhaps even burdening the program with poorer code. While this example might be overemphasized, it does illustrate how an attitude perpetuated more software maintenance complexity. High-level factors exist in many computer-based organizations and it is not likely that they will immediately disappear. It is safe to say, however, that attitudes developed in this environment need to be rectified if the maintenance process is to improve.

Another source of maintenance attitudes comes from a more fundamental area than the work place. These are the attitudes that grow out of the educational experience, and their roots lie deep within our educational behavioral patterns. Two of the important individual needs developed during this experience are those of creativity and skill growth. Much of the initial exposure to computer programming was through educational experiences that permitted the creation of programs as a method to learn new skills. As a result, little or no encounter with the maintenance process occurred except for the individual program debugging (and that task was consciously justified as part of the development scenario). Attitudes towards maintenance were not even realized at this stage. However, the attitudes that supported the theory that programming is a creative and skill-strengthening process flourished. Perhaps the occupational title of programmer as opposed to software engineer connotes the creative attitude as well.

When new programmers are exposed to real software maintenance situations, they are totally unprepared to handle the depth of the software maintenance process. They find themselves performing a programming task that is constrained by another style as well as design. In addition, they are confronted with the responsibility of understanding a program whose functions may be totally unfamiliar to them. It has been argued, however, that it is very possible that maintenance functions will provide new skill growth with programs that employ current software techniques.[3,5] Unfortunately, programmers are usually assigned maintenance on specific systems for some duration and in time they will achieve the maximum skill growth that can be derived from such a system. In addition, not every system employs new or current software techniques. Therefore the time it takes to outgrow the skills of the system may be very short.

The need for creativity and skill growth extends beyond the educational environment into the professional work place. When these individual needs are denied, the result is usually a search for a place where they can exist.[7] Recall that one of the effects of software maintenance is high turnover. It is from these concepts that the author suggests that a fundamental problem in the software maintenance process is the possible hindrance of creativity and professional skill growth. This hindrance may create low-level attitudes, as they will be termed, which include uncreativity and nongrowth. It is also possible that these low-level attitudes create a subconscious attitude in the programmer that manifests itself in the more recognizable high-level attitudes described earlier.

Recognition of where maintenance attitudes originate points towards a method of how they can be improved. Since the attitudes described in this paper stem from the professional work place and the educational environment, it is only natural that a method of improvement occur in these places. The author has provided suggestions for maintenance attitude improvement via the conventional educational techniques currently in use. These suggestions do not exclude the fact that there are probably other techniques available. However, the author would like to stress the fact that the source of any method for improvement must come from the professional work place and the educational environment.

SOLUTIONS IN THE EDUCATIONAL ENVIRONMENT

Formal education curriculums, which include programming courses, should also include courses that address software maintenance issues. There is also an important need to clarify what the occupational functions of a programmer entail. Secondary schools that offer programming classes to students might want to structure their course objectives to include exposure to the maintenance process. At this educational level it is not necessary to investigate the methodologies used in maintenance but rather to introduce the concepts of what maintenance is about.

At the college and university levels, students who are required to take data processing courses as part of a non-DP curriculum should also be exposed to the maintenance issues. As potential users of computer-based systems, they would develop attitudes that may eventually be part of a computer-based organization. If nothing else, the user community would at least be conscious of the complexities involved in the maintenance process.

Finally, and most important, are those college, university, and programming schools whose curriculums are designed for computer science and programming graduates. A software maintenance course or courses should be required as part of the requirements for graduation. The content of such a course can be divided into the three areas of maintenance management, maintenance programming, and maintenance attitudes. One of the major objectives of the course is the realization that software maintenance is essential for the success of the computer-based organization. It would be of little value to present the course in the way maintenance has been viewed in the past, that is, "It's a necessary evil."

The particular topics of a software maintenance course may include:

- the need for maintenance,
- what are maintenance functions,
- tools for maintenance,
- preventive maintenance,
- planning maintenance groups,
- interfacing with users and developers,
- change control procedures,
- monitoring maintenance activities,
- attitude strategy,
- integrating creativity,
- maintenance programming practices.

Many of these topics would seem theoretical to students who have never been involved in a computer-based organization. However, a very practical topic, which can almost be the course in itself, is that of maintenance programming practices. Choosing a language that is a curriculum requirement, an instructor can easily create assignments where students need to correct, adapt, or enhance a prewritten program or programs. In the correction assignment, students are given the task to find and correct a problem that was the result of a poor design or oversight in functional specifications. Two programs, one structured, the other unstructured, would be given to the students. The grade in such an assignment could be dependent on the success of corrective action, or method of implementation, that is, maintaining program design uniformity, documentation standards, and the time it would take to make the change. Of course, it may take several assignments before the time factor would be meaningful. In the adaptation and enhancement assignments, the instructor could use the corrective assignment programs and ask for implementation of new functions or modify some existing functions. Use of consistent programs might strengthen the maintenance concepts without frustrating the student, a possible side effect in such a course. If frustration becomes a barrier to completion of an assignment, then perhaps the instructor could alter the assignment to allow the student to find a technique for getting around the frustration.

Another topic area is the integration of creativity and skill growth within the maintenance process. Here students could learn how to channel creative energy in a direction that is beneficial to both the maintenance task and the individual need. Typically, this topic would concentrate on the enhancement aspects of the maintenance process.

Another course for software maintenance might be a study of maintenance programming techniques for specific program languages, for example, FORTRAN, COBOL, PL/1, and ASSEMBLY. In this course the concentration would be on the problems and solutions for correction, adaptation, and enhancement within the constraints of the language. An assignment that an instructor could give might be to modify two different language programs to perform a similar function. Again, many assignments can be created to support such a course objective. If a programming-languages maintenance course is not practical, then another alternative would be to include a maintenance section within the teaching of formal languages themselves. In such courses, students can learn the advantages of a programming language in the development process as well as the technique to be used with the language during the maintenance process.

As of this writing, the author has found little or no evidence that indicates that higher educational institutions are requiring maintenance courses as part of a computer science degree program. Maintenance courses, like those suggested, would help current and future programmers by raising their consciousness of the maintenance process and by improving their maintenance attitudes. It is hoped that the shock of maintenance functions, which new programmers encounter, will thereby disappear.

SOLUTIONS IN THE PROFESSIONAL WORKPLACE

Improving maintenance attitudes in the work place will be more difficult than resolution in the educational environment. Unfortunately, many programmers have less than positive attitudes already established. However, changing attitudes is definitely possible. One of the best ways to initiate an improvement is to have the computer-based organization demonstrate a positive recognition of the maintenance process. Programmers and managers involved with maintenance activities could be given extra incentives for their work. Such incentives could include extra compensation while assigned to maintenance groups, time compensation for extra or odd work hours, and advanced training in software techniques.[3] Another improvement method is the establishment of a planned development path for programmers. A typical path may include the rotation of maintenance assignments and development assignments on a regular schedule. This will allow programmers the opportunity to flex their creative skills and to support their need for professional growth. There is much evidence that indicates that some computer-based organizations are already gearing up for some type of attitude improvement strategy. One of these strategies is the use of outside training seminars in software maintenance for managers, project leaders, and group leaders.

Aside from commitments on the part of the computer-based organization, there is a need for individuals to examine their own methods for improving attitudes. Computer science is a technology that almost dictates change, and it is vital that individuals be conscious of those changes that are needed for professional development.[10] Professional programmers must commit themselves to improving attitudes about maintenance. Given the implementation of maintenance courses in the formal educational environment, programmers can attend those classes to improve maintenance skills and attitudes. After all, there seems to be that kind of commitment whenever a new software technique is introduced.

Attitude development in the professional work place is possible through a mutual effort on the part of the programmer and the computer-based organization. Perhaps it will require the educational community to react first before individuals and companies commit themselves to an attitude improvement plan.

CONCLUSION

Focusing the effort on improving attitudes towards maintenance will help in the development of the software maintenance process. Attitudes are complex, but their complexities can be more easily understood if the origins of these attitudes are examined. Recognizing the source of attitudes will foster new methodologies in attitude improvement that can thus become part of the programming process. No single solution to attitudes will result in attitude change. It will take time for attitude improvement theories to become acceptable as solutions to maintenance problems. The educational environment, which enabled the achievement of new software techniques, will also be inspirational in the development of new maintenance concepts. The future of software maintenance looks more promising with the recognition of underlying problems and implementation of new solutions.

REFERENCES

1. Clark, David M. "Maintenance Programming." *Computerworld,* 14 (1980), pp. 26–32.
2. Lientz, Bennet P., and E. Burton Swanson. *Software Maintenance Management.* Reading, Mass.: Addison Wesley, 1980.
3. Schwartz, Barbara. "Eight Myths About Software Maintenance," *Datamation* 28 (1982), pp. 125–128.
4. Borghesi, Nancy T., and Patricia L. Krapf. "Structured Methodology." ICP Interface. *Data Processing Management,* Spring 1982, pp. 37–42.
5. Reutter, J. "Maintenance is a Management Problem and a Programmer's Opportunity." *AFIPS Proceedings of the National Computer Conference* (Vol. 50), 1981, pp. 343–347.
6. Silverman, Robert E. *Psychology* (2nd ed.). New York: Meredith Corporation, 1974.
7. Beeler, Jeffry. "Exec Identifies Seven Reasons Why DPers Quit," *Computerworld,* 16 (April 19, 1982), p. 25.
8. Carlyle, Kim. "Programmer's Job Service Averages 18 Months," *Computerworld* 15 (1981), pp. 19–20.
9. Chapin, Ned. "Productivity in Software Maintenance," *AFIPS Proceedings of the National Computer Conference* (Vol. 50), 1981, pp. 349–352.
10. Weinberg, Gerald M. *The Psychology of Computer Programming.* New York: Van Nostrand Reinhold, 1971.

A methodology for minimizing maintenance costs

by LINDA BRICE and JOHN CONNELL
Los Alamos National Laboratory
Los Alamos, New Mexico

ABSTRACT

Research conducted in the case study of a large applications system shows that the two primary causes of high maintenance costs are

1. The frequency of user-requested changes to software
2. The psychological complexity of the software

A "tool kit" is suggested that, when applied to the design of new systems or rewrites, will

1. Produce systems that users are less likely to need changed
2. Contribute to the reduction of psychological complexity of code, making it easier to change when necessary

The tool kit is easy to use, can be applied to large or small systems in any language on any equipment, and requires no purchase of hardware or software.

INTRODUCTION

Maintenance costs escalate when software must be changed. Sometimes there are user-requested changes because the system does not meet the user's needs, and sometimes there are "bugs" because the systems and the individual program modules composing those systems are not well structured. All changes, whether necessary to fix bugs or desired to improve or add features, are difficult when program code is psychologically complex.

Quantifiable costs associated with software applications include the following: computer resources used by the application; programmer staff time plus computer-resource costs expended to maintain the application; and associated user time spent trying to learn and to use the end product.[1] The focus of this paper is on programmer staff time expended to maintain the application. Maintenance will be defined as all changes required to keep a system running according to the user's needs, including

- Corrections to programs necessitated by coding errors or misunderstanding of user requirements;
- Changes to programs required owing to changes in environment or legal/regulatory changes not under the control of the user;
- Enhancements or optimizations that alter the processing environment, often including minor new features.

Research performed in the case study of a large applications system has shown that the number of changes applied to a system and the psychological complexity (in particular, the misuse of branching instructions) of the code undergoing change both correlate positively with maintenance costs in terms of programmer effort.[2,5,19] The term *psychological complexity,* as used here, refers to those elements of programming style that make the resulting software difficult to maintain.

This paper is written to suggest several aids for the reduction of software maintenance costs. The first suggestion is for the data processing professional to employ certain metrics to estimate the expense of maintaining software. Software shown to be expensive to maintain may then be subjected to a break-even/payoff analysis for economic justification of a rewrite. When rewrites appear to be economically feasible, care must be taken so that the new system is indeed easier to maintain than the old.

Many data processing shops continue to maintain production systems, despite high maintenance efforts, simply because they work. It would be helpful to have a method of deciding just when psychological complexity contributes enough to the maintenance costs to be economically unfeasible. There comes a time when, because of psychological complexity due to poor initial program design, or due to many "patches," rewriting the program (or set of programs) is more economically justifiable than continuing to maintain it.

In order to develop new systems and rewrites of existing ones that will have lower maintenance costs, a methodology is needed for designing with future maintenance in mind. Because psychological complexity is causally related to maintenance costs, the methodology should provide a means for minimizing such complexity. Since it has been demonstrated that requests from the user for changes correlate significantly with maintenance costs, the methodology should also aim at maximizing user satisfaction with new systems and rewrites in order to reduce future service requests.

WHEN TO REWRITE

One method for deciding when to redesign existing computer applications involves deriving an economic break-even/payoff analysis using a five-step process:

1. Track maintenance costs for a time period and then project future costs, using straight-line trend analysis.
2. Measure the complexity of the existing code using a demonstrated metric.[3–5,20–24] This step does not contribute directly to the break-even/payoff analysis, but it does provide confidence that program complexity contributes to maintenance costs.
3. Estimate cost of rewrites.
4. Estimate costs for the maintenance of the new system after implementation.
5. Prepare a break-even/payoff analysis. In this projection (Figure 1), maintenance costs for the present system are shown as a straight line. Total cost for the proposed

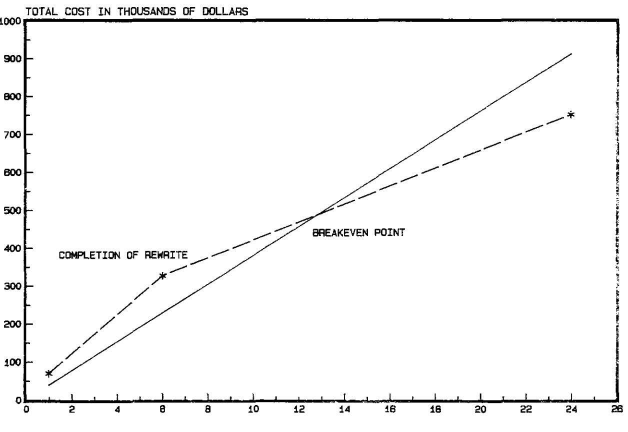


Figure 1—Time in months (assuming rewrite takes two people six months)

system is shown as a broken line with cost to completion of rewrite having a steep slope (it includes cost of maintaining the present system), and cost after completion of rewrite having a gentler slope since the new system will be easier to maintain.

TOOL KIT FOR REWRITE

The life of software systems is traditionally viewed as a cycle or sequence of iterative events. Recently, the life-cycle concept has come under fire.[6,7] This paper is not intended to pass judgment on the life-cycle concept—many versions exist, not all without merit. What is proposed here are a few techniques that we hope will reduce the costliness of maintenance. In order to describe the helpful tools, it is necessary to assume that the software to be maintained is written, not purchased, and that the development of that software proceeds in some order decreed by management. It is suggested that in order to minimize the number of post-implementation requests from users for changes, users be involved in setting objectives, and that production of output facsimiles and prototyping occur early in the development process.

The assumption will be made that software writers' management and the end users' management agree on the events that must take place to get the system up and running. Those events should be scheduled in a visual form (Gantt charts, Figure 2). The events will vary from project to project, but will necessarily include consultation with the user to describe system functions and software development. DP and user managements should meet before each major step to review the schedule.

The major goal is inexpensive maintenance. The tools are recommended (a) to force users' participation in the design, which will cause them to request fewer changes later, and (b) to produce lucid code that requires less effort per change. They are:

1. *System requirements definition (SRD).*
   Tool: Scheduling guideline
2. *System Design.*
   Results in system design document (SDD).
   Tools: Output facsimiles or prototypes
   Data flow diagrams (DFDs)
   Policy statements
   Data dictionaries
3. *Internal Design.*
   Results in Requirements Specification Package (RSP).
   Tools: DFD's of proposed system (from SDD)
   Approved output formats (from SDD)
   Policy statements (from SDD)
   Data dictionaries (completed from SDD)
   Logic-flow charts
   Program abstracts
   Program design walkthroughs

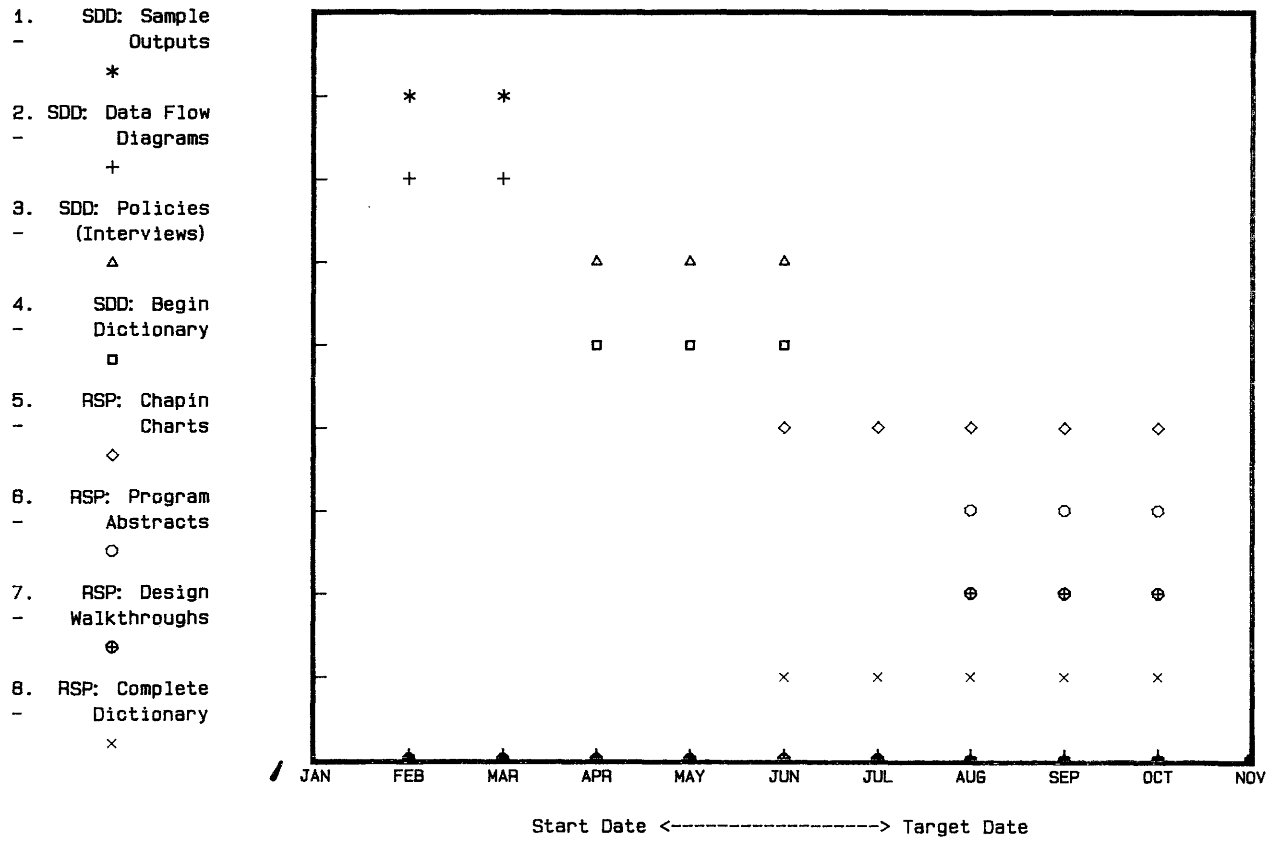


Figure 2—Gantt chart

The methods and tools mentioned do not depend on team makeup or on computer-based tools. None of the tools are original with this paper. What is proposed here is the integrated use of the tools to meet the stated goals.

Systems Requirements Definition

The systems requirement definition (SRD) will not be covered in depth in this paper because, except for the schedule, there are no specific tools recommended. The purpose of the SRD is to identify proposed objectives, define the project scope, define the organizational units involved, identify the end users, identify production or purchase approaches, and construct a rough schedule and cost/benefit for each alternative.

Cost/benefit analysis has already taken place when the system is a rewrite. It is inherent in the break-even/payoff analysis mentioned under "WHEN TO REWRITE." If the system is an entirely new development, the assumption is that a cost/benefit study would be necessary for a go/no-go decision by management at this point, prior to any actual development effort.

The schedule is not intended to be rigid as to dates. It is intended to identify the tasks to be performed, the parties involved, and the order in which the tasks will be performed. When reviews are held prior to the end of each phase (task), the remainder of the schedule can be reviewed and adjusted for reasonableness.

Gladden warns that "system objectives are more important than system requirements ... concentrating on objectives can go a long way to prevent a system from 'evolving' into one that the user does not want or need."[7] The life-cycle wheel model of system development, which concentrates on viewpoints, stresses that "... requirements analysis is viewed as a design activity from a user viewpoint. This design is synthesized from various (incomplete, inconsistent) user scenarios and other expressions of needs. The emphasis is on what functions the system is to perform, and how the system interacts with the users."[8]

The SRD is, then, the project's starting point and the place where objectives are defined.

System Design

The ultimate degree of user satisfaction with a new system or rewrite is often determined in the early stages of analysis and design. It is recommended that intensive interviews be conducted with the user during this phase. Such interviews should concentrate primarily on net outputs—the part of the system that will be visible to the user after implementation. Users may have little interest in how data will be massaged to produce these outputs.

System design: document components

Careful users will want to know how the accuracy of the information contained in the net outputs can be guaranteed. A system design document (SDD) should, therefore, contain the following elements:

1. A brief description of the framework within which the proposed system will operate, including
   a. constraints imposed by the operating environment
   b. required hardware/software configuration
   c. allowances for future contingencies.
2. Samples of proposed net outputs, such as report layouts and screens.
3. Proposed formats for net inputs, showing how data will be captured at original collection points.
4. Visual diagrams of data flows for the present system (either manual or automated) and the proposed system.
5. Policy statements giving a decision method for each procedure shown in the above diagram(s).
6. Rigorous definitions of all data elements shown in the above diagram(s).

System design: tasks

Development of the SDD components need not be undertaken in the order given above. The SDD's development guidelines may specify tasks to be performed in preparing such a document, and the order in which they should be performed.[9] The following is a brief description of each of these tasks.

System design: tasks—describe system environment. At this stage, the system designer can recognize when the new system will exist in a physical environment that may impose constraints on the design. The task during this phase should involve documenting the nature of that environment and identifying areas that might impose design constraints.

System design: tasks—describe net outputs. Examples of proposed outputs can be produced rapidly without using actual applications software. The editor on any system can be used to produce a text file that, when copied to the line printer, will produce a facsimile report or screen layout. The main advantage of this approach is that content and format can be changed easily without modifying software. In addition, the facsimile reports provide an immediate focal point for user interviews. Users who tend to be vague about system requirements can often be coaxed into being more specific by discussing information contained in the new outputs. If the output formats are approved by the user before the system design begins, the result should be fewer design changes and service requests after implementation.

If an installation has available the necessary tools (i.e., flexible database systems), it is strongly recommended that a prototype system be brought up at this early stage. "It is now recognized ... that although the customer may state his requirements very firmly at the beginning, his perception of the problem begins to change as he begins to consider how the solution development ... is proceeding."[10] Peters, Gladden, and McCracken and Jackson all recommend rapid prototyping to combat wholesale requirements changes.[6,7,10] The remainder of the SDD is charged with demonstrating that the approved sample net outputs can be produced accurately.

System design: tasks—describe net inputs. If the user is familiar with existing inputs, it is probably not necessary to produce samples. There may be, however, implications in the above components of the design for new methods of data capture or even entirely new data elements to be captured. In this case, it is important to solicit user approval of new input formats such as data entry screens. The method for providing examples of proposed input formats can be the same as that for output formats—sample forms produced with a text editor form an obvious, simple manner.

System design: tasks—produce data flow diagrams. For a visual representation of the flow of data between functions performed by a system, the use of the data flow diagram (DFD) is highly recommended. DFDs have been explained in Yourdon's structured analysis and design technique, and are described by Dr. Marco.[11] Basically, these diagrams consist of bubles, arrows, and parallel lines. The bubbles represent a procedure, the parallel lines represent a data store, and the arrows represent the flow of data between the procedures and data stores. The diagrams are ordered by degree of detail—the highest level (Level 0) contains only one bubble labeled with the system name and shows only net inputs to and outputs from the system (Figure 3). The lowest-level diagrams show elementary procedures and data elements (Figure 4). One suggestion for the number of descriptive levels is seven, plus or minus two. The rule also applies to the number of bubbles or procedures per level. The diagrams should remain visually digestible, since they are the tool for user interviews in this phase.

DFDs demonstrate for the user how net inputs will be transformed into net outputs; therefore they serve as a primary check on the accuracy and completeness of the outputs. This technique tends to minimize unnecessary or over-complex procedures and maximize user satisfaction.

Each of the bubbles or procedures shown in the lowest-level DFD should have an associated policy statement that describes the decision method proposed to perform the procedure. These policy statements should be expressed in structured English or pseudo-code so that they are unambiguous yet still intelligible to the user (Figure 5). They should be developed in the interviews with the user so that they are, in fact, the user's policies. Each policy statement should correspond to a bubble on a low-level DFD.

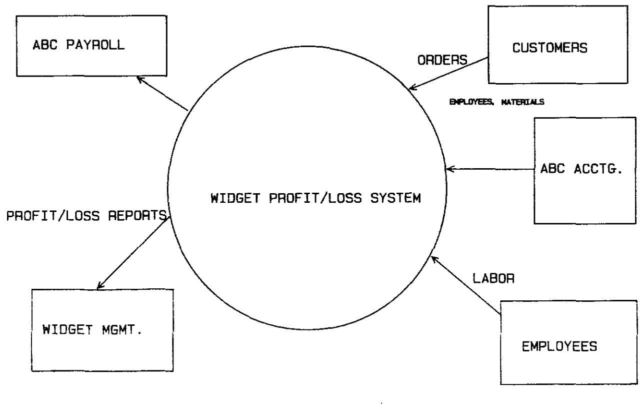


Figure 3—Level 0 data-flow diagram

These statements of user policy should eventually become online documentation for production source-code in the form of prologues (abstracts) for procedure modules. Initially, they serve as a guide to system design; later, they can serve as a maintenance aid.

System design: data dictionary

Each of the arrows in all of the levels of the DFDs will have a label. The SDD should include a "dictionary" defining each of these labels. The definition of a data label on a high-level diagram should be in terms of the labels on the diagram of the next lower level. At the lowest level, each label should also be defined as to how, when, and where that element will be captured.

If the dictionary is complete and rigorous, it serves as a proof that the user's requirements, as expressed in the policy statements, can be satisfied using the data defined therein. Each definition should correspond to the level of the DFD on which it can be found as the label of a data flow. This also answers the designer's question, "what data do I need, and where can I find it?"

Internal Design: Requirements Specification Package

Once the SDD has been approved by the user, "internal" design can begin. Here, internal design will only address those elements necessary to develop low-maintenance software. The requirements specification package (RSP) components will include

- Copies of the DFDs that identify program modules
- Approved output (reports)
- Data dictionary from the SDD
- Policy statements from the SDD
- Logic flow diagrams for each module (Chapin charts)
- Program abstracts.

The data dictionary may be revised during this phase of the project, and policy statements should contribute to the functions listed in the program abstract.

Internal design: Chapin charts

It is suggested that Chapin charts,[12] Nassi-Shneiderman Structured Flowcharts,[13] or the structured programming design method (SPDM)[14] be used to describe logic flow for each program module. The three are similar in philosophy, and any one can be used to bridge the gap between module need (basic requirements) identification and executable code. The document will be referred to here as a Chapin chart.

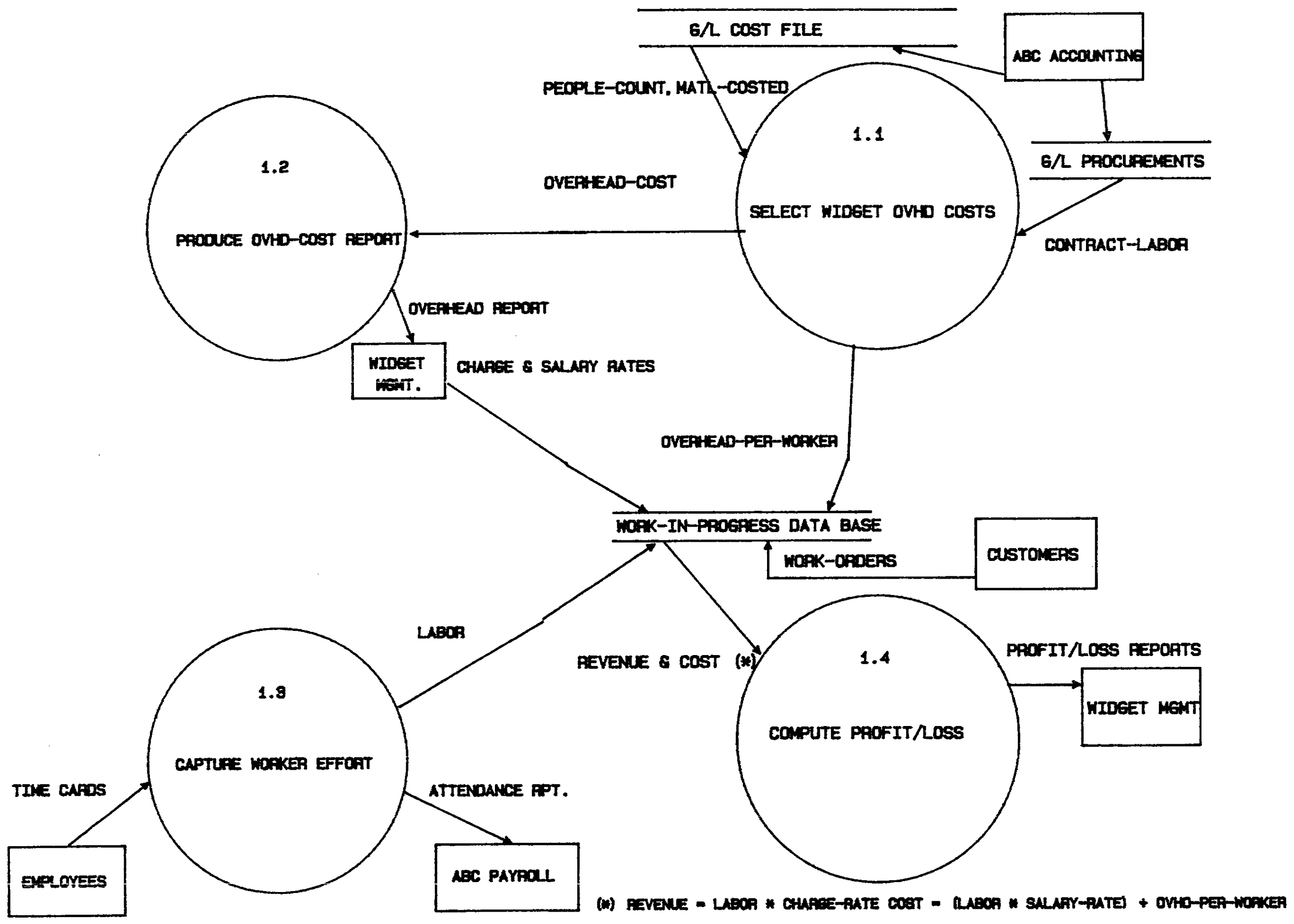


Figure 4—Level 1 data-flow diagram

The lowest-level DFDs in the proposed system section of the SDD represent processes in bubble format. Usually, each of these processes identifies a program module, as well as the inputs and outputs. Policy statements in pseudo-code or in structured English accompany the DFDs. The combination of inputs, outputs, and policy statements form the skeleton of a Chapin chart. If a database-management system is used, it will also have been defined in the SDD as "required software configuration" under the operating environment. If not, files or specific formats for data-transfer mechanisms must be specified prior to the construction of Chapin charts.

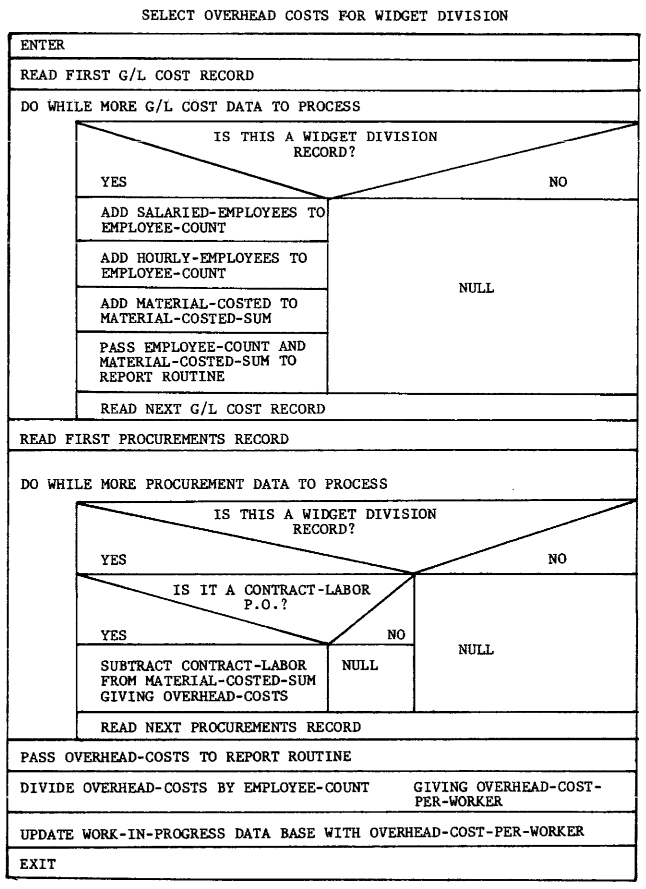
```
    For each ABC Company General Ledger cost record
pertaining to Widget Division:

         Add salaried employees and hourly employees to
         employee count;

         Add material costed to material-costed sum;

         Pass employee count and material-costed sum to 1.2.

    For each ABC Company Procurement record pertaining to
Widget Division:

         Subtract from material-costed sum those purchase
         orders involving contract labor, resulting in
         overhead costs.

         Pass overhead costs to 1.2 for use in management
         report.

    Divide overhead costs by employee count, resulting in
overhead-cost-per-worker.

    Update the work-in-progress data base with overhead-
cost-per-worker.
```

Figure 5—Sample policy statement

The Chapin chart is based on this cumulative knowledge, sometimes with the addition of special processing algorithms. The reader is referred to the references for in-depth explanations of this logic-flow chart.[12,13,14,15] The method, in essence, consists of visually representing a set of program building blocks that allow single entry/exit and strictly limit branching, a practice known to increase psychological intelligibility. The set of program structures includes SEQUENCE, IFTHENELSE, DOWHILE, DOUNTIL, and CASE. When used properly, the set of combined structures lends itself to a well-structured program guide where arbitrary transfers of control are impossible. Figure 6 is an example of a Chapin chart.

The benefits of the Chapin charts are

- Provision of a "GOTO-less" map to be translated directly into a programming language



```
SELECT OVERHEAD COSTS FOR WIDGET DIVISION
ENTER
READ FIRST G/L COST RECORD
DO WHILE MORE G/L COST DATA TO PROCESS
    IS THIS A WIDGET DIVISION RECORD?
    YES
        ADD SALARIED-EMPLOYEES TO EMPLOYEE-COUNT
        ADD HOURLY-EMPLOYEES TO EMPLOYEE-COUNT
        ADD MATERIAL-COSTED TO MATERIAL-COSTED-SUM
        PASS EMPLOYEE-COUNT AND MATERIAL-COSTED-SUM TO REPORT ROUTINE
    NO
        NULL
    READ NEXT G/L COST RECORD
READ FIRST PROCUREMENTS RECORD
DO WHILE MORE PROCUREMENT DATA TO PROCESS
    IS THIS A WIDGET DIVISION RECORD?
    YES
        IS IT A CONTRACT-LABOR P.O.?
        YES
            SUBTRACT CONTRACT-LABOR FROM MATERIAL-COSTED-SUM GIVING OVERHEAD-COSTS
        NO
            NULL
    NO
        NULL
    READ NEXT PROCUREMENTS RECORD
PASS OVERHEAD-COSTS TO REPORT ROUTINE
DIVIDE OVERHEAD-COSTS BY EMPLOYEE-COUNT    GIVING OVERHEAD-COST-PER-WORKER
UPDATE WORK-IN-PROGRESS DATA BASE WITH OVERHEAD-COST-PER-WORKER
EXIT
```

Figure 6—Chapin chart

- Provision of a document that graphically depicts logic for the purpose of review (peer review, team walkthrough)
- Provision of a test-bed guide.[15]

It has been noted that Chapin charts are not devices that provide functional hierarchy, interfaces, or data flow.[14] The contention here is that there is no necessity that Chapin charts respond to those needs, since they are met by the DFD. What Chapin charts do well is control flow of executable code within a higher-level functional design. This toolkit provides the functional design via DFDs.

Internal design: walkthroughs

Approved DFDs showing processes (program modules), inputs, outputs, policy statements, functional hierarchies, interfaces, and data flows are available from the SDD phase; program-module logic design is graphically represented by the Chapin charts. Because of the importance of structuring program code for understandability and readability in the maintenance phase ("good" structure equals psychologically clear code and minimum branching), the Chapin charts should be subjected to a peer review before the coding phase. The review should not only insure the structure of the individual modules, but should double-check to see that elements are defined in the data dictionary, that the process will accurately perform what was intended in the higher-level diagrams, that the outputs conform to the early prototype specifications, and that a program abstract is present. The abstract would minimally consist of

- Purpose
- Input (arguments/files/other)
- Output (arguments/files/other)
- Functions (10 or less)
- Local variables
- Subprograms called
- Errors (fatal/non-fatal)
- Standards violations.[16]

An example of a program abstract is in Figure 7. The purpose of walkthroughs is improved (low-maintenance) quality of the product. The value of walkthroughs shows up ultimately in the maintenance phase. "The inspection process shifts the discovery and correction of errors and defects from software's operational period to the early design stages. Since the cost for software corrections during operations is many times the cost incurred in detecting problems during design, inspections provide an unusual leveraging of cost/benefit over the entire life cycle of the software."[17] Although a heavy commitment is necessary for the time of team members and moderator participation, other benefits beyond low-maintenance code are

```
PROGRAM NUMBER:                 56-311

SYSTEM DESIGN NUMBER:           56

DATA FLOW DIAGRAM NUMBER:       1.1

PROGRAM NAME:             SELECT OVERHEAD COSTS FOR WIDGET DIVISION

AUTHOR:                   C. G. POND

PURPOSE:                  DETERMINE OVERHEAD COSTS FOR WIDGET DIVISION

INPUT:     1.    GENERAL LEDGER COST FILE
                 (EXTERNAL FILE NAME = D70129A)

           2.    GENERAL LEDGER PROCUREMENTS FILE
                 (EXTERNAL FILE NAME = D70101A)

OUTPUT:    1.    OVERHEAD COST REPORT
           2.    UPDATED WORK-IN-PROGRESS DATA BASE

FUNCTIONS:       1.    FOR EACH ABC COMPANY GENERAL LEDGER COST FOR
                       THE WIDGET DIVISION, EXTRACT EMPLOYEE COUNTS
                       AND COST OF MATERIAL TO DATE.

                 2.    REDUCE THE MATERIAL COST BY THE AMOUNT OF
                       CONTRACT LABOR.

                 3.    CALL A SUBROUTINE TO PRODUCE AN OVERHEAD
                       COST REPORT, PASSING THE EMPLOYEE-COUNT, THE
                       ORIGINAL MATERIAL-COST, AND THE MATERIAL
                       COST REDUCED BY CONTRACT LABOR.

                 4.    UPDATE THE WORK-IN-PROGRESS DATA BASE WITH
                       OVERHEAD COST PER WORKER. (REDUCED MATERIAL
                       COST DIVIDED BY NUMBER OF EMPLOYEES.)

LOCAL VARIABLES:          MATERIAL-COSTED-SUM
                          OVERHEAD-COSTS
                          EMPLOYEE-COUNT
                          OVERHEAD-COST-PER-WORKER

SUBPROGRAMS CALLED:      1.    56-312
                               PRODUCE OVERHEAD-COST REPORT

COMPILATION OPTIONS = COBOL5, EL=T, LO.

ERRORS:   NONE

STANDARDS VIOLATIONS:    NONE
```

Figure 7—Program abstract

accrued, such as "training and exchange of technical information among the programmers and analysts who participate in the walkthrough."[18]

CONCLUSION

Use of this tool kit will not guarantee that the resulting system contains minimal psychological complexity and maximized user satisfaction. It is possible to misuse the tools. The intention of this paper was to explain some of the factors that cause software to be expensive to maintain, and to provide aids that may be useful in designing low-maintenance systems.

REFERENCES

1. Brice, L. "Existing Computer Applications—Maintain or Redesign: How to Decide?." *Proceedings of the 1981 Computer Measurement Group International Conference*, pp. 20–28.
2. Brice, L., J. Connell, and J. Taylor. "Deriving Metrics for Relating Complexity Measures to Software Maintenance Costs." *Proceedings of the 1982 Computer Measurement Group International Conference*. Phoenix, Ariz.: Computer Measurement Group, Inc., 1982, pp. 134–141.
3. Halstead, M. H. *Elements of Software Science*. New York: Elsevier North-Holland, 1977.
4. McCabe, T. J. "A Complexity Measure." *IEEE Transactions on Software Engineering*, SE-2 (1972), 1, pp. 308–320.
5. Connell, J., and L. Brice. "Complexity Measures Applied to an Applications Case Study." *Fourth International Conference on Computer Capacity Management Proceedings*, 1982, pp. 121–128.
6. McCracken, D. D., and M. A. Jackson. "Life Cycle Concept Considered Harmful." *Software Engineering Notes*, 7 (1982), pp. 29–32.
7. Gladden, G. R. "Stop the Life Cycle, I Want To Get Off." *Software Engineering Notes*, 7 (1982), 2, pp. 35–39.
8. Yamamoto, Y., R. V. Morris, C. Hartsough, and E. D. Callender. "The Role of Requirements Analysis in the System Life Cycle." *AFIPS, Proceedings of the National Computer Conference* Vol. 51, 1982, pp. 381–387.
9. Brice, L., and F. Welch. *Manual of Procedures and Standards*. Administrative Data Processing Division, Los Alamos National Laboratory, Los Alamos, N.M., 1982.
10. Peters, L. "Relating Software Requirements and Design." *ACM Proceedings of the Software Quality and Assurance Workshop. Software Engineering Notes of the ACM*, 3 (1978), pp. 67–71.
11. DeMarco, T. *Structured Analysis and System Specification*. New York: Yourdon, Inc., 1978.
12. Chapin, N. "New Format for Flowcharts." *Software Practice and Experiences*, 4 (1974), 4, pp. 341–357.
13. Nassi, I., and B. Shneiderman. "Flowchart Techniques for Structured Programming." *SIGPLAN Notices of the ACM*, 8 (1973), 8, pp. 12–26.
14. Marca, D. "A Method for Specifying Structured Programs." *Software Engineering Notes of the ACM*, 4 (1979), 3, pp. 22–31.
15. Yoder, C. M., and M. L. Schrag. "Nassi-Shneiderman Charts—An Alternative to Flowcharts for Design." *Software Engineering Notes of the ACM*, (1978), 5, pp. 79–86.
16. Control Data Corporation. *Final Report of the Aircraft Noise Prediction Program Phase II*, (Contract No. NAS1-13983), NASA, Langley Research Center, Hampton, Va., July 1978.
17. Werner, F. L. "Software Inspections: Process and Payoffs." *Computerworld*, April 12, 1982.
18. Yourdon, E. *Structured Walkthroughs*. New York: Yourdon, Inc., 1978.
19. Chrysler, E. "Some Basic Determinants of Computer Programming Producitivty." *Communications of the ACM*, 21 (1978), pp. 472–483.
20. Chapin, N. "A Measure of Software Complexity." *AFIPS, Proceedings of the National Computer Conference*, Vol. 48, 1979, pp. 995–1002.
21. Zolnowski, J., and D. Simmons, "Measuring Program Complexity in a COBOL Environment." *AFIPS, Proceedings of the National Computer Conference*, Vol. 49, 1980, pp. 757–766.
22. McTap, J. "The Complexity of an Individual Program." *AFIPS, Proceedings of the National Computer Conference*, Vol. 49, 1980, pp. 767–771.
23. Berlinger, E. "An Information Theory Based Complexity Measure." *AFIPS, Proceedings of the National Computer Conference*, Vol. 49, 1980, pp. 773–779.
24. Zolnowski, J. "Taking the Measure of Program Complexity." *AFIPS, Proceedings of the National Computer Conference*, Vol. 49, 1980, pp. 329–336.

Quality assurance and maintenance application systems

by BARBARA J. TAUTE
Time Inc.
New York, New York

ABSTRACT

Modifications to application systems in production can have a devastating effect on the environment if the changes are not handled correctly. A comprehensive quality assurance (QA) approach can help minimize this potentially harmful effect. This approach involves all groups: users, data processing center, applications programming, and quality assurance.

The QA approach should address four areas:

1. Phased approach
2. Procedure flows
3. Maintenance guidelines
4. Implementation

This paper describes the QA phased approach successfully developed at Time Inc. The phased approach consists of the definition and implementation of eight phases, envisioned in a circular life cycle. Emergency processing is considered separately. Procedure flows consist of diagrams and charts listing responsibilities of the participants. Maintenance guidelines contain helpful hints and checklists and provide direction to the participants.

The benefits are noteworthy: The phased QA approach consolidates groups, forms a standard for maintenance procedures, and increases productivity.

INTRODUCTION

Computer application software systems in production exist in a volatile environment. An environment is unstable when data, hardware, systems software, and usage are in flux. Environmental changes will probably create a need for modification to the application system itself.

Production application systems are in high-risk environments simply because they are in use. The user community expects and needs correctness to meet deadlines (in contrast to a system in development, which is not yet in the users' hands). Any change in a system can have an adverse effect on its operation.

This paper will address a methodology for understanding, controlling, and benefiting the maintenance environment. The environment is best perceived as affecting four groups: users, data processing center, applications programming, and quality assurance. This approach document, developed by the Quality Assurance Department of Time Inc., continually reflects the responsibilities of all four groups. Only through a combined coordinated approach encompassing the four groups can changes in production application systems be controlled and made manageable.

TIME INC. ENVIRONMENT

Time Inc., with headquarters in New York, is a diversified company encompassing publishing, forest products, and video. The Application Programming Department supports the development and maintenance of applications for the departments of Magazines, Books, SAMI (Selling Areas—Marketing, Inc.) and Corporate Staff offices. It is composed of 240 people, split mainly between Chicago and New York. A liaison role of user of record provides an application interface to the user department. The Quality Assurance Department provides methodologies and structured approaches to all departments.

Because the people in applications programming were of varied backgrounds (multiple groups), disciplines (software packages to in-house development), experience levels (1 to 15 years), and sophistication (hard coded assembler to test data generators) a need was perceived for a methodology to consolidate these efforts. This methodology was to be created by the quality assurance team, which possessed an independent, experienced, state-of-the-art view for programming structure.

The methodology was to address several areas. The first area was to pertain to maintenance systems, the next to development systems, the third to testing, the fourth to measurement of all processes, and the fifth to productivity. This paper deals with the first approach, that of maintenance systems, which is defined as follows.

DEFINITIONS OF MAINTENANCE

A maintenance application system is defined as being in production and in use by the user groups. Since maintenance (back end) was being organized first (before a development approach), the quality assurance department was not able to define a specific set of test, documentation, operation, recovery, or security requirements that would need to be met before a system was officially considered to be in maintenance. This sort of requirement would compose the turnover criteria from a development effort. Since no standard existed, programs showed varying levels of completeness in their production environment. The original design criteria no longer existed, and in many cases system documentation had not been kept up to date. Yet these systems were providing accurate, meaningful data and output results to the user departments. In order to include all systems in the approach, it was necessary to define *maintenance* as pertaining to any system in a production environment.

The Quality Assurance Department, however, was able to influence the definition of a maintenance life cycle. It is defined to consist of eight structured, related phases with definitive criteria and responsibilities shared by four groups (users, data center, application programming, and quality assurance).

MAINTENANCE STRUCTURED PHASES

The maintenance life cycle (Figure 1) is represented as a circle with one phase leading into the next. Its eight phases refer to the following:

1. Request phase—An expressed desire for a change to a system
2. Estimate phase—A calculation of effort to complete the change
3. Schedule phase—An identifiable release date planned for the change
4. Programming phase—The modifications to a controlled source copy of the system
5. Test phase—The verification that the change performs as expected
6. Documentation phase—The modification to system, user, and run specifications
7. Release phase—The replacement of the old system with the changed system
8. Operation phase—The day-to-day usage of the system

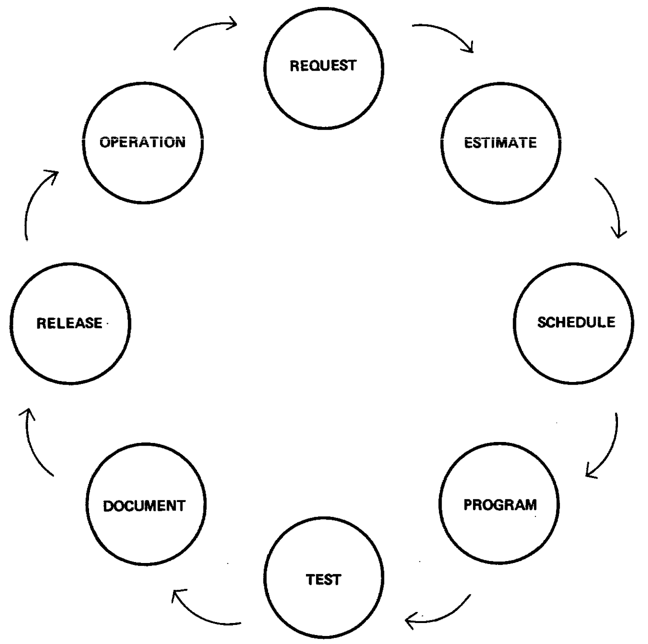


Figure 1—Maintenance life cycle

Request Phase Definition

A request for a change, whatever its form, initiates a process that will affect an existing production system. A change request is a request for an investment, since there will be time and therefore money spent by various groups. A change request may occur for numerous reasons. If there are no change requests, the environment is stable; and if a system is not used, no changes are necessary. A system in production will by definition have changes made to it, because it is being used. The reason for a system change is often a function of the time a system is in use and is not peculiar to one system or another nor an indication of how poorly or well it was developed.

Since changes can come from so many different sources (users, data processing center, programmer, manager, auditor), it is necessary to have a standardized format representing requests. It is not possible to evaluate changes or to assign priority without established criteria. A change request form helps pinpoint symptoms of a problem, level of need, and time of involvement; and appropriate levels of approval can be expressed and retained.

In the environment at Time Inc., several change request forms were being used. The Quality Assurance Department consolidated the best portions of these forms to devise a new form for all groups.

The completed change request form specifies the deliverable for this first phase. When it is completed, it is transmitted to the librarian. This transmittal initiates the next phase, the estimate phase.

Estimate Phase Definition

The estimate phase has its own significant steps. Upon receipt of the change request form, the librarian assigns it a unique number. Then, if the change request concerns an application system error, the suspect error needs to be verified prior to estimation. It is hoped that this will help eliminate reported problems that are the result of nonadherence to operational procedure, user misunderstanding, or invalid data. Next, a person from Applications Programming needs to evaluate the change request and estimate the number of hours it will take to correct the problem. The project manager must always review this estimate for concurrence. The estimate must be as thorough and as accurate as possible, since this number will be used as the basis for a scheduling process. The estimate is also reviewed with the user of record.

The process of internal reviews helps to eliminate incorrect estimates and permits several individuals to learn from the estimation process, rather than just a few. At Time Inc., levels of review established were based on fixed hours. These limits were refined as upper management modified their level of review requirements and as personnel became more confident in the estimate process.

The completion of the estimate and review portions (including signatures on the change request form) determines the end of the estimate phase. The schedule phase follows.

Schedule Phase Definition

Functional enhancements compose a significant portion of the maintenance workload. In order to handle these changes as efficiently as possible from the standpoint of all departments, a scheduled release concept is used. Releases of systems are numbered and given predetermined dates based on their business cycle, such as every 60 or 90 days. Nonemergency changes are then assigned to a specific release number. Each release contains multiple changes and forms a new replacement system. The advantages of this process are

1. Consolidation of changes
2. Increased stability of the production system
3. Reduction of training requirements
4. Better planned and managed workloads

The change requests provide the basis for a project tracking system. A summary entry for each outstanding change forms the agenda for a meeting of all four departments. Periodical sessions, called Change Control Board meetings, are held to discuss the various needs and priorities of groups. After the meeting the project tracking system and the change request form can be updated with a scheduled date and release number.

When a scheduled release accommodates multiple changes, it represents many added hours of effort. There is an increase in the risk of malfunction for that release. Therefore an approval process similar to the review process in the estimate phase must be followed. This process helps guarantee that management will maintain proper awareness of releases of a predefined magnitude.

At Time Inc. the process of scheduled release was a new one. There was initial skepticism about its success until one group installed this process, with close guidance. The process was an overwhelming success: Users were pleased to have expected dates for changes, requests were dealt with and not put aside because of time commitments, and application programming was favorable to fewer releases in number. A 3-month release cycle was initially established, but this was ultimately changed to a 2-month cycle to be more responsive to the natural business cycle.

The assignment of a release date and time frame for a change marks the completion of the schedule phase. The next phase is the programming phase.

Programming Phase Definition

Modifications of application systems in production differ from changes in systems in development, since production systems are currently operating as an integral part of the user's business. It is essential to insure that modifications are being made to the current operational version of a system. Thus, a procedure to control the source should be followed. The program phase consists of creating a test version of the controlled production source code of the system, making the programming changes according to the approved design, and initiating the test process and updating the documentation. Associated version numbers and dates will help guarantee that the proper source code files are being modified. A program log at the start of each program will help to identify the changes and ease the next modifier's job. While modifying the test version, care must be taken to maintain the integrity of the production system. Files can be lost or data can be extensively altered if the integrity of the systems is compromised. In addition, the programmer must be cautioned to make only authorized changes. When a segment of code is opened up, it is not the time to make a "nice little fix." These fixes can be disastrous (in error), can cause extra test time (invalidating estimates), or can just be undesirable (unneeded enhancement) to the users.

At Time Inc. various existing support systems (SLIM, PANVALET) required the retention of varying source copies. An attempt to consolidate these was not deemed advisable, because of varying form as well as location (varying cities, varying locations). Therefore, all groups agreed upon a minimum number of storage repositories and upon adherence to defined control procedures. A long-range paperless repository is currently being planned.

At the conclusion of the programming phase, a set of program and system changes exist. This marks the beginning of the test phase.

Test Phase Definition

Testing of a production system is one of the most critical and important phases of the entire change process. Testing helps insure that the replacement system will function properly and not disrupt the user environment. The quality of testing is a function of both the thoroughness of the test plan and the quality of the test data. The test plan should be comprehensive and should include unit testing, integration, and system testing of the changed elements and their interfaces. Good test data start with a good test base, which is kept current as production systems change. If a test base does not exist, *create one.* Regression testing (verifying changed systems run correctly with known data) can only be accomplished through maintaining a controlled test set.

The Program Manager has the ultimate responsibility for the correctness of system changes, but other groups may also become involved. Often users will test for functionality, and the data center may verify via parallel runs. This involvement can provide further confidence in system integrity.

At Time Inc., through the initiation of this methodology, the programmers were made aware of the importance of keeping a good test base to achieve time savings, cost savings, and accuracy.

The completion of the test phase is evidenced by correct operation of the changed program and by a test approval signature on the change form. Prior to a system going live (after a correct test run) documents must be updated. The next phase discusses the documentation issues.

Documentation Phase Definition

Three forms of documentation are considered for maintenance systems: system, user, and run.

Since documentation must be accomplished before the system is released to production, it is therefore represented in the life cycle immediately before the release phase.

System documentation is maintained to help a programmer learn a system and its elements. This information describes the system in its past phases (history) and its present state and is used as the basis for future changes in the system. These items, at a minimum, should be continuously maintained: high-level system flow, system functional description, self-documented program source compilation listing, and documentation describing data flow. System documentation should be kept in an accessible but secured library area with a checkout procedure. The documentation should be retrieved by the programmer or analyst, updated, and replaced in the library. The amount and level of detail for controlled documentation must be maintainable, since obsolete, incorrect documentation is more confusing than no documentation at all.

User documentation should be composed of user manuals, error lists, and functional descriptions for the use of the system. A change in a system may very well change the way the system appears to the user. An increase in functions can increase the capabilities of the system. Remember: If users do not know how to use the system, they cannot use it and will not use it. Any change in a system should be evaluated for the possible necessity of correcting or updating user documentation. These modifications to the user documentation should be made before the system is released.

Run documentation is the combination of materials (Job-Procedures, JCL, restart/recovery instructions, etc.) required by the data processing center to operate the system. Therefore, the minimal requirements need to be set by the data center. Often a system change will result in some modification in the way the system is run. This means that existing operating instructions should be changed at the time of release. If the data processing centers have standards or procedures, these should be followed for form and content of run information.

Reviews by the project manager or systems analyst can help guarantee that all documentation is complete. The user of record should help determine if user documentation is thorough and clear.

Training may not be required for relatively small system changes, but it is certainly necessary for larger ones. If there is a functional change, users need to be retaught how to use the function. New operating procedures may need to be taught to the data processing center. The type and amount of training is very system-dependent. The development of good documentation can facilitate this training process.

At Time Inc., documentation levels varied considerably among applications programming groups. Some groups had required comprehensive documentation, and very detailed user manuals were found to exist. For others, the manuals were out of date and had not been maintained. One data center had existing standards; for another they were still in development. The introduction of this methodology helped to standardize all documentation.

The completion of the documentation phase is evidenced by signoffs on the change request form and in the tangible documents themselves. The subsequent phase is the release phase.

Release Phase Definition

The release phase is the natural last step of the change process as well as the most critical step in the life cycle of a change. It is at this time that users become excited about the increased functionality that exists, programming is enthusiastic about their system release, analysts become concerned about the unforeseen impacts of the change, and the data processing center looks for a clean, easy installation of the changes. This critical period can be greatly eased by a well-controlled and well-communicated release procedure. A prerelease review conducted by the Quality Assurance Department with the participation of all four groups can ease this process. Quality assurance should verify the following:

1. System installation readiness
2. Adequate system testing
3. Adequate approval testing
4. Completeness of documentation
5. Installation requirements specified
6. Transmittal form supplied

Once these characteristics are verified, the system can be considered ready for release to production. These criteria should be presented punctually for review so that the review does not postpone the release process. After the review, the system is officially turned over to the data processing center, and the user is informed that the system is ready for use.

A controlled release process insures the definitive release of systems. In the absence of this process, systems may never officially be turned over to production but continue to be run as test systems. This can lead to multiple versions and confuse users. At Time Inc. this controlled process eliminated some of these redundancies and solidified the release process.

The turnover to production marks the completion of the release phase. This is signaled by the running of the production jobs by the new system, and also officially by the signatures on the change release transmittal. The last phase is the operation phase.

Operation Phase Definition

Operation is the day-to-day activities of any system. Even though a system is stable, various elements can cause the erosion of a system solely through its use. Not all these elements can be planned for or controlled because of the vast, unknown combinations of actions that can occur. What is helpful, however, is to plan for an orderly description of the event if an error does occur. Users and the data processing center must be trained and educated about the functions and use of the system. The more operational aids written into and about the system, the more closely the environment can be handled by the users and the data processing center, and the less programmer assistance will be necessary. Testing should be planned. It is costly, but remember that thorough testing can improve the systems operation.

After the system has gone through one complete production cycle, the four groups should convene and conduct a postrelease review, led by the project manager.

A system in operation runs until it is outdated, is too difficult and costly to support, no longer functionally serves its purpose, or needs extensive changes. A change leads back to the request phase; a need for a rewrite leads into a new system development process.

As in most companies, Time Inc. has numerous systems in production and several in development. This process helped all groups by controlling the maintenance process and defining its components.

A phase included separately is emergency processing. Because it is treated in a unique fashion, the process should be discussed by itself.

EMERGENCY PROCESSING

Every attempt should be made to keep emergency releases to a bare minimum. In fact, the by-product of a well-controlled maintenance cycle ought to be the practical elimination of emergency releases. However, they still exist in the real world and therefore must be planned for. An emergency is a change of such importance or impact it must be considered immediately and out of the normal maintenance structure. Emergency processing expressed for the eight phases is described below.

The emergency request phase can be originated by a change request form or a "midnight" phone call. A change request form is completed as soon as possible to record all work.

The emergency estimate phase follows the standard structured process if the emergency change does not need an immediate fix. The programmer or analyst must immediately evaluate the change, if it is a highly critical fix, and proceed directly to program and test. This can apply to both a temporary and a permanent fix. If the correction cannot be accomplished rapidly, a manager's approval is necessary. If the permanent fix must be made during extended working hours, the same phases should be followed; but the time can be compressed and the modifications can be installed outside the maintenance release.

The schedule phase for emergency problems falls outside the scheduled release concept. The schedule and time will depend on the complexity of the change.

The program phase is no different for emergency problems. A source still needs to be extracted from a library and modifications made to it.

The test phase is perhaps more important for emergency problems. Since a bad situation already exists, extreme care must be taken to make sure the fix improves the situation rather than aggravating it. For changes to be installed immediately, all groups involved should approve the testing prior to its release.

The emergency documentation phase must be conducted in the same manner. However, emergency changes may require different timing. Temporary documentation may need to be developed before the permanent updates are released.

The emergency release phase will be under control of the data processing center once the proper approvals are received. The transmittal remains the same.

The emergency operation phase follows the same guidelines once the emergency fix is in place.

Thus, the description of the phases and the emergency phase give an understanding of the cyclical nature of maintenance systems.

PROCEDURE FLOWS

A visual representation of the phases of maintenance is best provided through diagrams and responsibility charts. The format as shown in Figure 2 is used to describe visually the functions of the participants. The action column briefly describes the event in English text. This event is represented by a symbol, such as a listing, tape, or paper form, which is placed beside the heading of the participant. Lines drawn connecting the symbols indicate multiple participant input or output. In this way all interfaces are shown in a clearly serialized fashion. A chart should be constructed for each phase of the maintenance approach. The level of detail is optional, depending on levels of need for any one particular group or desired summary level for another. This form of phase representation clearly identifies each participant group's activities and their interface groups. The second portion shows the responsibilities of the participants. Each title is listed and English text used to delineate their activities during each maintenance phase. A set of procedure flows was constructed for Time Inc.'s environment.

GUIDELINES

Guidelines provide procedures for handling various aspects of the maintenance process. These guidelines should be somewhat flexible in order to encompass subsequent suggestions or considerations for the maintenance process. They should not be hard-and-fast or demanding rules. As with any guidelines, they are not meant to be a replacement for good judgment. In many cases checklists are provided to help participants increase their level of understanding about their responsibility for the phase. In other cases, helpful hints and known areas of concern are identified for the participants. A set of guidelines for Time Inc.'s environment was constructed.

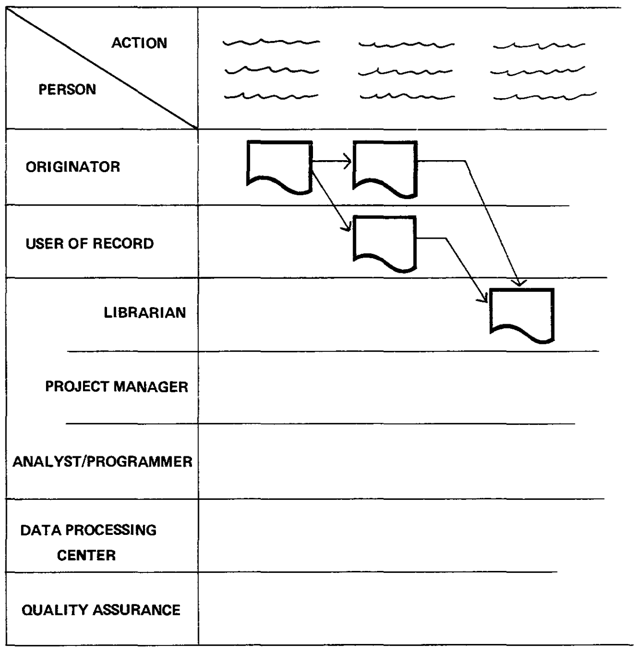


Figure 2—Procedure flows

IMPLEMENTATION

For a methodology to be successful, planned action and attention must be given to its installation. At Time Inc. a separate document was produced that addressed the implementation of the maintenance approach. Before the approach was implemented, program control library, automated tracking system, move to production procedures, and required system documentation were addressed.

The implementation document itself had the following chapters:

1. Introduction
2. Installing the Procedure
3. Training the Participants
4. Monitoring the Effort
5. General Installation Schedule

SUMMARY

A controlled, phased maintenance approach can simplify and assist in the delivery of timely, correct versions of applications systems software. This process is defined as a circular life cycle with the collaboration of four groups to insure its success.

Time Inc. has defined a methodology for its environment and is in the process of a successful installation and implementation of the procedures.

Human investment techniques for effective software maintenance

by NICHOLAS L. MARSELOS
Western Electric Company
Lisle, Illinois

ABSTRACT

This paper presents methods for improving the maintenance of software by addressing the psychological issues that impact on software maintenance personnel. The emphasis in this paper is on making software maintenance developers more effective through goal setting, by using team-building approaches, through support personnel, and by using skill profiles to plan for their technical growth.

INTRODUCTION

Today everyone recognizes the problems in software maintenance. Over half of the people now developing software are involved in maintaining it. This absorbs a great deal of the energy and creativity that we have in our software-development community. With increasing emphasis the question is being asked, "What can be done to solve this problem?"

Unfortunately, the difficulties in software maintenance are not the result of a single problem, but of many. Many single solutions have been employed to make software-maintenance developers more effective. Most of these have been technical solutions that help the software-maintenance developer design or program or control the software product more effectively. There has, however, been a real lack of emphasis on the human investment in software maintenance. Although the technical solutions are beneficial and important, the real gains in improving the productivity and effectiveness of software-maintenance developers are attained by improving their motivation. This paper focuses on the psychological issues that plague software-maintenance developers. It offers multifaceted solutions that, when applied, will improve their motivation and provide many other benefits to their companies.

MAINTENANCE IS A MULTIFACETED ACTIVITY

The technical nature of software maintenance is a well-understood problem. The software-maintenance developer performs a variety of activities. These include defect correction, feature addition, and working with users. The software maintenance developer must also keep current on the system and support environment in which his software product operates in order to change the product as required by changes made in the environment.

The software maintenance developer must constantly use some of his creative energy to understand and get around the constraints of the software product he or she is maintaining. This product is usually poorly documented and in many cases written in an unstructured and very difficult to change manner. The software maintenance developer must also spend part of his or her time dealing with the user in a variety of roles that may include trainer, consultant, and complaint handler. All of this is usually done with time pressures resulting from very short development schedules.

These parts of the software-maintenance developer's job are well understood, and many solutions have been offered to help in these areas. But there is an entire area in which very few solutions have been offered, and which creates more significant problems for the software-maintenance developer.

Psychological Issues of Software Maintenance

The software-maintenance developer is in a position with a great deal of psychological pressure. This pressure comes from a variety of sources. One source is the feeling of being "stuck" in the job of maintaining a specific software product. Often developers feel that management is indifferent to their problems. The management passes down the message "Don't make waves, just get the job done." The management's attitude may be reflected in their lack of interest in the education and personal growth of the software-maintenance developers. This makes software-maintenance developers feel like second-class citizens in the organization.

The evolution of software maintenance has resulted in an environment of independent islands. Each island supports one or a few software-maintenance developers working on their specific product, unattached and uninvolved with much of the rest of the development organization. In this environment, the software-maintenance developer feels unsupported and that the job rests solely on his or her shoulders.

The psychological pressures on software-maintenance developers has a demotivating effect. Over a period of time, their productivity begins to diminish and this has bad effects on the cost and quality of the software products being maintained.

Administrative Problems

The administrative problems of software maintenance as seen by management revolve around the issue of keeping cost to a minimum. This cost directly correlates with the number of people involved in maintaining the software products. Management is faced with the problem of optimizing to the minimum number of software-maintenance personnel that can keep software products maintained in a healthy way. However, this often creates the alienation and morale-lowering feelings experienced by software-maintenance developers.

The rotation of software-maintenance developers to different assignments and the ability to provide backup for them is often a serious administrative problem. This problem stems from the long-learning curve required by the software maintenance developer to effectively maintain the software product. The lack of the effective documentation, which is common to most software products, is a major contributor to the problem. In addition, the skeleton forces often put on maintenance projects makes it virtually impossible for maintenance personnel to back each other up on systems effectively.

MULTIFACETED SOLUTIONS TO THE MAINTENANCE PROBLEMS

The overall effect of the administrative problems in software maintenance is to demoralize the software-maintenance developer and to frustrate management. These problems have a negative impact on the software product, which over a period of time begins to degenerate. Software management sees a large investment slowly dwindling away with little control on their part to effectively reverse the process. The software-maintenance developers feel pressed in the position of too little support and too few resources to do the job effectively.

A key factor in improving the software-maintenance developers' productivity and quality rests in motivating those developers. An environment must be established in which the software-maintenance developers feel supported and can proceed to do their job in the most effective way. This takes a commitment by management. It also requires a cultural change that is first initiated by defining the organizational objectives, and then enhanced by creating a supportive environment that meets the psychological needs of the software-maintenance developers.

Goal-Setting for the Organization

One of the most important needs in any organization is to have well-defined goals in which everyone in the organization supports. Software maintenance has its own set of goals. These goals can conflict with each other, as indicated in the experiment by Weinberg.[1] Table I identifies the goals and the negative effects of those goals.

Software-maintenance goals must be set, then ranked to avoid conflicting goals. As Table I shows, setting the goal of timeliness, that is getting the products out on time, may jeopardize the efficiency and maintainability of the software products being developed. This is true whether the products are in the initial development stage or in the maintenance-development stage of their life cycle. Table I also shows that an organization both placing emphasis on timeliness and demanding high maintainability puts tremendous pressure on the software-maintenance developer, since these two goals are in direct conflict with each other. It is therefore important not only to establish the goal or goals that the organization would like to achieve but also to rank those goals to minimize conflicts.

TABLE I—Goals of Software Maintenance

Goal	Negative Effects
Timeliness	Inefficiency in operation
	Jeopardize maintainability
Operational efficiency	Increased development costs
	May not be user friendly
Customer satisfaction	Development longer
	Greater resources required
Minimum costs	Schedules shortened
	Better project planning required
	User satisfaction can suffer
Improve maintainability	Schedules longer
	More resources required
	More emphasis on documentation
	More emphasis on design

The process of establishing goals for the organization has to begin with management. The management should sit down and decide what goals they really seek, whether it is to have customer satisfaction or improved maintainability or timeliness of its products. The goals may be global to the organization or modified for each project. Once the goals are selected, they should be ordered in terms of priority, the most important, the next most important, and so on. Once this list is created, it should be checked to make sure that conflicting items aren't adjacent. If they are, the management must then decide which is the most critical item of the two and move the conflicting item down on the priority list. Finally, the entire management must support these goals.

After the goals are well defined and written, everyone in the organization must be informed what the goals are, and their support must be promoted. This can be done by having meetings of management and the software-maintenance developers. These meetings should be used to stress the importance of these goals to the organization and to encourage suggestions on how best to achieve the goals.

The success with which these goals will be accepted depends on whether achievement of these goals is rewarded. The rewards can be of two forms. First, there must be recognition awards. These are publicized awards for software-maintenance developers who achieve the goals established by the organization. For example, if timeliness is a specific goal, then those developers that bring their products out on time would receive recognition for their accomplishments either in the organization's news bulletin or in memoranda. Second, there should be substantive awards given for achievement of the goals. These include pay raises and promotions or any other form of substantive award. In all cases, it should be made common knowledge that the award has direct correlation with achievement of the goals set by the organization.

When goals are established and promoted in such a way, they will have a profound effect on the cultural aspects of the organization. The software-maintenance developers will be more motivated once they have clearly in mind what the organization considers important in the performance of their activities. This will give the software-maintenance developers objectives to direct their energies toward. It will not, however, set up the type of supportive environment that is necessary for them to be effective in achieving their goals. For that, a more supportive team environment must be created.

Instituting Team Consciousness

Most software-maintenance developers are put in a situation where they have few people to rely on for support. New project developments often have several team members working in concert to complete the project. When the project is in its maintenance cycle, only a skeleton force, usually just a single individual, will maintain the project. The software-maintenance developer is left without having the benefit of sounding boards or other people's expertise to resolve problems. From a management perspective, this is necessary to keep the cost of maintenance low. There are, however, ways to solve this problem without substantially adding to the number of software-maintenance developers. This can be done by creating teams that perform specialized functions which support the software maintenance developer.

Augment Groups

Augment groups are informal teams chartered to propose solutions to problems or to suggest improvements for the organization. They are founded on the concepts employed in quality circles now popular in Japan. They differ from quality circles in that they are populated by professionals and that they can apply to any specific goal that the organization seeks.

The charter of an augment group is implied by its name. The name and charter of the group should be chosen so as to be directed to a specific organizational goal. For example, there could be productivity groups or user-relationship groups or more-effective-documentation groups. Each group would be chartered to look into problems relating to one specific area.

The process of organizing and conducting these groups is now well defined in the literature pertaining to quality circles.[2] The groups should be formed on a volunteer basis. Software-maintenance developers should join the groups that most interest them. Several groups can be established within the organization. In the case of multiple groups, a facilitator or coordinator should attend the group meetings to ensure that there is no significant overlap in the activities or the solutions of the groups.

Augment groups in the area of software maintenance have to perform two vital functions. First, they offer an opportunity for some of the organization's problems to be addressed in a creative fashion; second, they give the software-maintenance developers an opportunity to participate in a team effort. The first benefit is one for the entire organization. Very often augment groups suggest substantial improvements that save the organization money or improve the productivity or quality of its products. The second benefit of augment groups is to provide a vehicle for the exchange of information among software maintenance developers. This gives them a feeling of team involvement even though their normal daily job might isolate them.

Peer Review Units

Two of the major problems facing software-maintenance developers are the lack of support in ensuring that a product they have developed is of good quality and will work effectively and second, the lack of backup personnel to relieve the load in pressure situations. Both of these problems can be relieved by the use of peer review units. A peer review unit is a group designated to review the work of the members within that group. The product can be the design documentation, the analysis documentation, or the program code itself. The group has a permanent membership. The membership is chosen to achieve the compatability conducive to fostering "egoless," or defensiveless, participation as described by Weinberg.[3] The review process could be conducted as a structured walkthrough. The review should have some formality, as described by Yourdon.[4] At the minimum, it should include a signoff sheet to indicate that the reviewers have accepted the product being reviewed.

The value of peer review units is in the improved quality of the program products being developed. The units also provide a backup situation by having the reviewers learn about the different systems that are involved. The results of field trials indicate that peer ratings of programs can be productively nonthreatening, and serve as incentives for programmers to produce higher quality code.[5]

Application Area Groups

Another type of group that can help in the communication and information and the support of software-maintenance developers is the application-area group. These are groups such as user-interface groups, or maintenance-developer groups, or birds-of-a-feather-type groups that work on problems or exchange information that is of common interest to the group. The group meets periodically to discuss either common problems or solutions, or to present new features or ideas, or to be used as a sounding board for activities or plans within the organization.

These types of groups are the easiest for an organization to establish. They are conducted as simple and informal sessions. They offer the software-maintenance developers an opportunity to exchange information and experiences. If users are involved, it provides a mechanism in which the software-maintenance developers and the users can improve their relations by gaining a better understanding of each others' problems and points of view.

Improving the Effectiveness of the Maintenance Personnel

The most significant qualification of software-maintenance personnel is that they have the technical capability to perform their job. This requires a well-planned training program. Another very important aspect of training is that the software-maintenance developer feel they are personally growing in their technical expertise and are keeping up with the state of the art of their profession; (the substantially high need for growth by software professionals was shown by Cougar and Zanacki[6]). Both of these are also important to the organization. Obviously, if the software-maintenance developers cannot cope with the technical aspects of their job, the product they are maintaining will suffer. If on the other hand, the software-maintenance developers have a strong desire to grow technically and that desire isn't met by an effective program for personal growth, then their morale becomes low and their productivity is negatively affected.

The training of software-maintenance developers must be a well-planned and controlled activity. The first step is to understand the skills that the software maintenance developers already have. This can be done by developing a profile of the skills they have acquired through education and on-the-job experiences. Next, the organization should profile the skills necessary for various jobs. Finally, the organization should profile the educational or training vehicles that can teach those skills. With this information, it's possible to develop an educational plan. This plan can be used for the rotation of software-maintenance personnel and also for planning their personal growth and training in a direction that benefits them and is suitable for the organization's needs.

Support Personnel

Many of the activities of the software-maintenance developer are of a low-level clerical nature. These are repetitive and time-consuming activities that diminish the software-maintenance developers' overall effectiveness. Many of these jobs, however, can be delegated to support personnel.

Software support personnel benefit the software-maintenance developer. They perform functions at various levels depending on their own abilities and expertise. These can range from simple data-entry jobs to performing the software testing. A more skilled helper can provide the first-line interface with users by handling some of the simple operations and questions needed to support users.

The organization will also benefit from having software support personnel. First, it reduces the cost of software maintenance by making the software-maintenance developers more effective. It allows the software maintenance developers to have more time for their specific activities and relegates the clerical support work to a lower-salaried individual. The presence of software support personnel also enforces a certain level of documentation: the documentation needed to help them do their job. Second, they provide continuity for the organization when new software-maintenance developers begin to maintain the product. Finally, a software assistant can be shared by several software-maintenance developers, providing added cost savings to the organization.

Software-maintenance developers can also be helped by expert technical support. This can be instituted by circulating a list of the persons in the organization who are most knowledgable about specific areas, for example, job-control language or program languages. The function of technical support experts is to answer inquiries about specific problems in their particular areas of expertise. This can be done by setting up expert tables where maybe for an hour or two every day the expert would sit and field all questions.

The use of technical experts within an organization works if the organization actively supports the policy of getting the job done in the most expedient and effective way. This means that the organization must discourage the not-invented-here syndrome and must encourage software-maintenance developers to use their innovativeness to meet the organizations' stated goals and not simply waste their energies on problems where they lack expertise. Rewards must be established for both the technical support person and for those who seek his or her help.

CONCLUSION

Software-maintenance developers often find themselves in an unrewarding and stress-filled job. They may feel ignored by their management and alienated from the rest of the organization. They may suffer from not having a full understanding of the goals of the organization and from the lack of support personnel to help them do their job effectively. They may have the feeling that they are stuck in their jobs and limited in their professional growth.

This paper describes several approaches that can be applied to making software maintenance developers more effective. The approaches deal with the psychological aspects of software-maintenance developers. The emphasis is on making them feel more a part of the organization and giving them more effective support. By using these approaches, the software-maintenance developer's motivation can be improved remarkably. Only when this is done is it possible to gain the maximum benefits from the technical tools and techniques for improving software maintenance.

REFERENCES

1. Weinberg, G. M., "The Psychology of Improved Programming Performance," *Datamation,* November 1972.
2. "Quality Circles in EDP," *System Development,* July 1982, Vol. 2, Number 5.
3. Weinberg, G. M. *The Psychology of Computer Programming* (1st Ed.). London: Van Nostrand Reinhold, 1971.
4. Yourdon, E. *Structured Walkthroughs* (2nd Ed.). New York: YOURDON Inc., 1970.
5. Schneiderman, B. *Software Psychology* (1st Ed.). Cambridge, Mass.: Winthrop Publishers, 1980.
6. Cougar, J. D., and R. A. Zanacki. "What Motivates DP Professionals?" *Datamation,* September 1978, pp. 116–123.

Structured software maintenance

by G. R. EUGENIA SCHNEIDER
Naval Weapons Center
China Lake, California

ABSTRACT

Many books are written about structured design and programming, but never about structured maintenance. True structured maintenance comprises four functional roles, called the manager, librarian, archivist, and programmer. The manager manages. The archivist protects contents of computer files and stores information about these files in an archive library. The librarian organizes and stores software documentation in the form of a program documentation package. The programmer, of course, programs, using versions of the archive and library documents with slightly altered contents, and records day-to-day activities in the programmer's notebook. A special tool used by programmers is emergency takeover, which is a procedure for taking maintenance control of a new program.

INTRODUCTION

In the universe generally depicted in the literature, software is carefully designed, written, tested, documented, used for a time, and then replaced by more up-to-date software created in the same way. Articles, books, and training programs have proliferated in support of this idealized environment. Unfortunately, no one seems to write books for maintainers of ancient, unstructured, undocumented software. Indeed, few will even admit that such persons exist.

This paper attempts to bridge the gap, to provide procedures and guidelines for practitioners in this much-maligned and neglected area. It presents an overview of a comprehensive system for structured software maintenance. The complete system is outlined in the data-flow diagram in Figure 1. The four sections of the diagram are for the four functional areas of structured software maintenance, and the abbreviations along the information flow lines are record-keeping and documentary tools used by the maintenance staff.

MEMBERS OF THE MAINTENANCE STAFF

Introduction

Because of the general disrespect for software maintenance, programming is often the only activity management sees as an appropriate maintenance function. So, no time or resources are allocated to many other crucial activities. In a fully-staffed maintenance group, resources must be assigned to roles called

1. Manager—The manager sets priorities for maintenance tasks, makes task assignments, and reports to higher management levels.
2. Archivist—The archivist keeps track of current contents of computer files and maintains back-up and retrieval procedures.
3. Librarian—The librarian organizes storage and retrieval of documents and other records about the software being maintained.
4. Programmer—The programmer trouble-shoots software failures, designs and tests program updates, and documents maintenance projects.

It is worthwhile to delineate the duties of each title separately (see Table I). The roles can be modified later if necessary to accommodate staffing limitations in a particular organization.

Maintenance Manager

The primary role of the maintenance manager is usually that of master psychologist because this person must conduct group therapy sessions to keep users, customers, operators, and programmers speaking to one another.

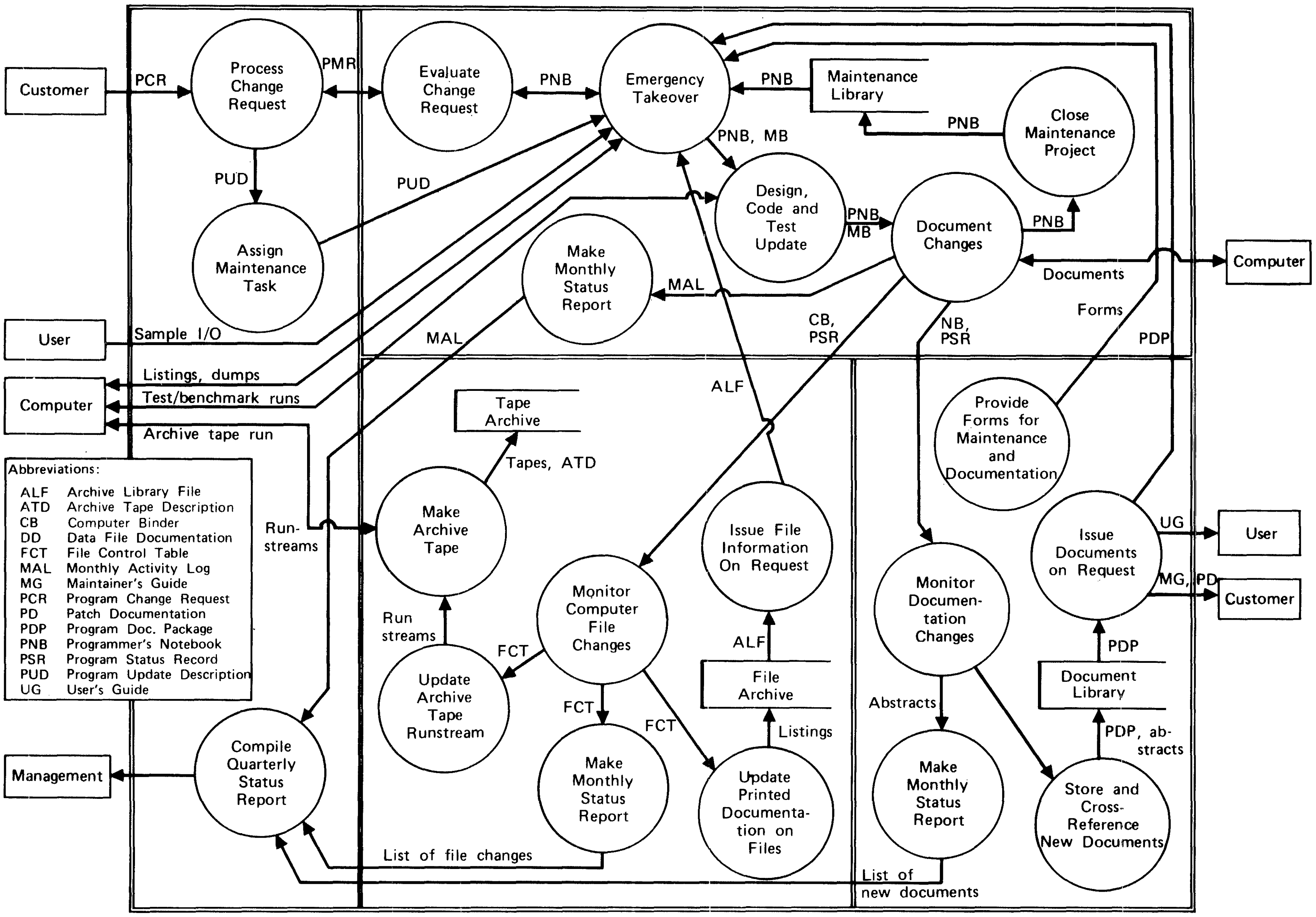
Officially, however, the most important duty of the maintenance manager is to monitor incoming program-change requests and to keep some kind of centralized trouble log. When a program change is proposed, the manager arranges for a timely decision on the criticality and feasibility of the proposal.

Table I—Software maintenance functions

	Who Does It?*				How often?†			
Activity	Mgr.	Arch.	Libr.	Prog.	Int.	Wk.	Mon.	Qtr.
Task Area								
Set/review priority	x						x	
Assign tasks	x					x		
Report to mgmt.	x							x
Make archive tapes		x						x
Monitor file updates		x				x		
Store file lists		x					x	
Retrieve file lists		x			x			
Store documents			x			x		
Retrieve documents			x		x			
Forms management			x			x		
Word processing control			x			x		
Distribute doc. updates			x					x
Update programs				x	x			
Update documentation				x	x			
Record-Keeping								
Status report	x							x
Archive library folder		x		x				
Document library file			x	x	x			
Programmer's notebook				x	x			

*Who does it? Mgr.—Maintenance manager; Arch.—Archivist; Libr.—Document librarian; Prog.—Maintenance programmer.

†How often? Int.—Intermittent, as needed; Wk.—At least weekly, better daily; Mon.—At least monthly; Qtr.—Once each quarter; (no entry—may never be done).



Abbreviations:

ALF	Archive Library File
ATD	Archive Tape Description
CB	Computer Binder
DD	Data File Documentation
FCT	File Control Table
MAL	Monthly Activity Log
MG	Maintainer's Guide
PCR	Program Change Request
PD	Patch Documentation
PDP	Program Doc. Package
PNB	Programmer's Notebook
PSR	Program Status Record
PUD	Program Update Description
UG	User's Guide

Figure 1—Data-flow diagram of the structured software maintenance system

Finally, it may be useful to have the maintenance manager develop and maintain the company's disaster plan. This may not be particularly appropriate as a maintenance function, but experience has shown that, somehow, the maintenance shop is tacitly held responsible when the system crashes.[1]

Archivist

The archivist is tasked with keeping up-to-date records of the contents of computer files. It doesn't matter what is in the files. The only criterion for inclusion in the archive library is that the information be in computer-readable form.

The archivist maintains a documentary folder for each computer library (a library being defined here as a logically connected set of separately accessible files). This folder includes

1. Current listings of the elements contained in the library
2. Methods for retrieving the information contained in the library at each step of its evolution, starting from the first time it was known to the maintenance group.

Note that the archivist does not keep track of which files are logically associated with particular programs or with each other. The only concern is with maintenance and protection of the physical files.

The file-protection task is chiefly accomplished by periodically generating back-up tapes, including all files that have been changed since the last time such a tape was written. When a computer file is updated in between the generation of these formal archive tapes, the archivist backs it up immediately in such a way that the change can be reinstated if the updated file is lost or damaged.

Document Librarian

The major function of the document librarian is to systematize storage and retrieval of documents and other written information of use to the maintenance staff. Then, at the end of each quarter, the librarian prints a list of all documents acquired and generated during the last quarter and distributes the list to the rest of the maintenance staff.

Two other tasks, however, routinely become assigned to a document librarian. The first is forms management—storing master copies of all forms used in the shop and seeing to their copying and distribution. The second peripheral duty is as word-processing manager for the maintenance team: storing skeleton copies of typical documents on the computer. A documentor, then, need only copy the proper skeleton and fill in the blanks. This has two advantages: it improves motivation of documentors to produce the assigned reports, and it assures that all documents of the same type will have the same format.

Maintenance Programmer

The operational cycle of maintenance programming is on the order of: make a change, make a run, curse, scribble, and loop.

Maintenance programming, however, is not the only task a programmer must attend to if the maintenance facility is to function effectively. There are three major types of informal documentation that must be generated by the maintenance programmer:

1. Programmer's notebook
2. Maintenance binder
3. Appropriately structured code, enhanced with in-line or in-code documentation.

This paper does not discuss the in-code or the in-line documentation since these topics have already been extensively covered. The emphasis here is on the external documents—the programmer's notebook and the maintenance binder. These are productivity tools that aid the programmer in producing high-quality software updates. They are not produced after the fact; they must be updated daily. The use of these tools is described in the next section.

RECORD-KEEPING IN A MAINTENANCE SHOP

Introduction

Many companies fail to recognize documentation as a crucial part of effective software maintenance. In a well-run maintenance group, as much as 75% of the time is spent in activities that can be loosely referred to as documenting.[2] But most of this activity does not result in formal reports so it often is not recognized as documentation. Some of these less formal documentary formats are the following:

1. Archive library file—A folder containing listings that define the contents of a computer library in sufficient detail so that the library's present or past contents can be retrieved. The archivist maintains these records.
2. Program documentation package—A binder containing a standardized set of section dividers that hold semi-formal bits of information about a program and its interactions with other software and hardware. The document librarian keeps these packages up-to-date.
3. Programmer's notebook—A collection of daily notes on computer runs, file updates, significant conversations, meetings, and so on. As is obvious from its name, this is the responsibility of the maintenance programmer.

These record-keeping tools constitute complete and efficient, if informal, documentation of the maintenance function. They record lessons learned while programs are modified, and they pave the way for any formal reports that are later required.

Before detailing these formats, a word about terminology is in order. In this paper, there was a need to use words that have both common and "computerese" meanings. The reader is asked to assume that if any ambiguous term (e.g., file, library, record) is used without some qualifier that indicates a computer-readable entity, the common English meaning is intended.

The Archive Library

Within the archive library, documentary folders are filed primarily by the computer on which they reside, then alphabetically by name. Tables of contents for the library are at the front of the folder, with the most recent listing first. The first section contains file lists, categorized as source code, runstreams, data, or text. Listings are stored in alphabetical order by file name and then by revision date, with the most recent on top. Another section contains information about using stored runstreams for documenting the library contents. Another holds listings showing how the files are retrieved from back-up tapes, again with the most recent first. Another possible division is for compile-link listings if executable program code is stored in the library. Finally, there is the omnipresent category of "Other."

The format of an archive library folder is shown in Table II.

Table II—The format of an archive library folder

Table of Contents: Most recent table of contents of the library usually including, for each element: name, date when it was last changed, size in words or sectors, and so on
Absolute elements: Compile-link lists for programs for which executable copies are stored in the library
Current element listings:
  Source lists—program symbolic elements
  Runstreams—sets of control (JCL) commands, that are accessed and executed (by the OS) as a unit
  Data—sets of data records that are accessed and used (by a program) as a unit
  Text—a catch-all category for all other elements
File Maintenance Records: Listings of runstreams used to document the library contents and information on how to retrieve the files from an archive tape
Other: Essential in ANY folder, no matter what its purpose

This is how the folders are organized in an actual archive library. For a programmer, however, the information may not be accessible in this form. The best solution for a programmer is to copy all files that relate to one program into a single library. Then the format is useful to both operatives. Some programmers decide instead to put all pertinent files in one archive library folder, regardless of where they are found on the machine. This is very frustrating if the group later acquires an archivist.

The Document Library

The document library contains whatever information is available, on paper, that is of interest to the maintenance team. The library is organized around a book of abstracts,[3] each of which references one of the programs, data files, procedures, and so on, being maintained. Documents in the library are organized first by system, a system being a group of programs and procedures recognized as a logical unit by the users. The next lower category is a subsystem, a name invented by programmers to identify a group of programs on a single computer or performing a single function. Finally, there are binders for individual programs and sometimes for individual data files.

There can never be a complete list of the documents that might be written about a program. As a start, the librarian stores information in a program documentation package, as shown in Table III. Whenever the contents under "Other" begin to overwhelm the rest of the package, it is time to be creative, to build a more specific set of dividers and to insert a sheet under "Other" to describe what was done.

Table III shows how program documentation looks in a formal document library. The format is designed, however, so that if there is no librarian, only programmers, the program documentation package is transformed into a maintenance binder. Though it has somewhat less formal contents, such as tape dumps, the format is the same. If no formal documentation exists in some binder sections when the program first appears for maintenance, the maintainer may write some as a matter of course. Others will never be produced formally; they will be represented by notes, if at all.

Daily Record-keeping Tools

Everything that happens during a maintenance task must be recorded in the programmer's notebook: code changes and their effects, program runs and their outcomes, insights gained, information gleaned, summaries of conversations, milestones achieved, and so on, *and* the date on which the event occurred. This notebook must have the feature that, although the first entries are made now, the records can be straightforwardly extended backward in time as more information becomes available. The programmer's notebook is a daily hand-written log of everything the programmer does, learns, hears, or acquires that relates to the task at hand.

Another important record-keeping tool, the maintenance binder, is quite formal by comparison. Recall that it is often synonymous with the program documentation package in the document library. When information is acquired that fits logically in the maintenance binder (i.e., it might be used someday, almost as is, in a formal report), it should be stored there. The programmer must present an up-to-date maintenance binder to the document librarian at the end of a maintenance

Table III—The format of a program documentation package

Abstract: Formal definition, one page long[3]
User's Guide:
Analyst's Manual: (a design document might go here)
Maintenance Information: Patch or maintenance document[3]
Source Lists and Cross-References:
Data Formats: Formal data file definitions[4]
Benchmark Inputs and Outputs:
Runstreams: "Canned" command files used in maintenance
Related Software: Any available information about how the program interacts with other software that is significant in the system (e.g., the program that creates a file it will read, or one that uses a file it will write)
Other: Anything that doesn't fit in another category

project. This binder is frequently the only documentation in the universe for that program. So, the structure of the maintenance binder is (entirely by design) the skeleton on which to hang any documentary information that might later be included in a formal report.

EMERGENCY TAKEOVER

The previous section described the record-keeping activities that support maintenance on a long-term basis and that were portrayed as a flow diagram in Figure 1. But what if there is no long-term basis for maintenance as for instance when a program is transferred to the maintenance team for the first time?

Emergency takeover is the name given to the steps involved in learning enough about a program to update it. Then, depending on the maintenance manager's decision, events proceed fairly evenly toward either a permanent program revision or a temporary patch. Since emergency takeover rarely appears in the literature, some time will be taken to explain the steps.

Suppose someone walks into the maintenance shop, and says, "XYZ didn't work last night... Fix it!" Assuming that no one in the group has ever heard of XYZ, there is a procedure to follow before beginning to make changes to the program. The steps are the following:

1. *Talk to the customer and the users.* If the program has never been heard of by your group, sit down and talk to the person who requested the maintenance task. Then talk to any users you can find. Ask what the program does, how to find a complete set of inputs and outputs, under what circumstances the program is used, if there is any user documentation, and so on.
2. *Search the document library.* If there is any information about XYZ in the library, find out who last worked on it and get the programmer's notebook and maintenance binder. Otherwise, have the librarian open a new file for the program and issue a set of dividers for creating its maintenance binder.
3. *Search the archive library.* Locate tables of contents for all pertinent program, runstream, and data files in the archive library. If you do not already have them in a maintenance binder, print listings and cross-references of the program modules. If the archive library has no information on the program, prepare an archive package as soon as you locate the program on the computer so that the archivist can take over maintenance of the files.
4. *Start a programmer's notebook.* Obtain and fill in whatever forms are used to head a programmer's notebook. From this point on, document in the notebook everything that happens during the maintenance project.
5. *Assemble sample inputs and outputs.* Find or create a complete set of sample inputs and outputs, including raw dumps (i.e., octal or hexadecimal, with no cleanup, decoding, or translation) of all data media (e.g., tapes, disk files) used by the program. Note: if the program is being changed because it aborted, the run that failed is needed to test the fix; but you also need benchmark data, a data set that ran successfully BEFORE the failure.
6. *Write the program update description.* This is a form defining the programmer's concept of the requested modification. The completed form is sent to the requestor for comment, via the maintenance manager. Its function is to ensure that everyone—the programmer, the manager, and the customer—can agree on exactly what the proposed change will accomplish, what resources will be expended, and so on. Once it is approved, the program update description becomes the nearest thing to a software specification that the maintenance shop is likely to encounter.
7. *Work backwards to fill in the programmer's notebook.* Using whatever information was gleaned from file lists, sample I/O, and so on, work backwards from the present to fill in the programmer's notebook. (Begin notes for each month on a new page in case information surfaces later that must be incorporated in this history.)
8. *Start to design the update.* When the program update description form comes back, approved or amended, it is time to stop the book work and start on the update. Translate the program update description steps into high-level pseudocode (sometimes called structured English) or into a module flowchart. You will be amazed at how much you already know about the program after these few, simple steps.

LAST WORD

When is a maintenance task finished? The answer has the same parameters as the answer to a related question: When is a newly developed program ready for production? Curtly stated, a program is ready when a test run, using the predefined benchmark data, has been approved by the programmer and the customer. When the librarian and archivist have also closed their files and a post-implementation review has passed without new changes being required, then the maintenance project is truly finished.

The purpose of this paper is to bridge the gap between the theoretical ideals expressed in the literature—structured design and programming, tactical and strategic planning, configuration management, formal documentation, and so on—and the real world we know, where decrepit, undocumented programs are being maintained. Until all software is well structured, reliable, maintainable, and documented, the methods outlined here will help maintenance practitioners to survive and succeed in an unfriendly universe.

ACKNOWLEDGMENTS

This paper is a condensation of a Master's thesis presented to California State University, Chico, in fall 1981. The thesis, in turn, was an expansion on an article for Auerbach Publishers. My thanks to both organizations for their help.

REFERENCES

1. Keston, R. *How to Develop an Effective Long-Range Data Processing Plan.* Rockville, Md.: Keston Associates, 1978.
2. Lientz, B. P., E. B. Swanson, and G. E. Tompkins. "Characteristics of Application Software Maintenance." *Communications of the Association for Computing Machinery* 21 (1978), 6, pp. 466–471.
3. Schneider, G., D. French, and L. Lucas. "How to Document Software," Naval Weapons Center CCF-87, August 1977.
4. Schneider, G.. "Format Guidelines for DRMM's," DRMM-74-9, 1974.

Application maintenance: One shop's experience and organization

by ROBERT E. MARSH
Dow Corning Corporation
Midland, Michigan

ABSTRACT

Several years of data on software support activity at Dow Corning are analyzed to illustrate the problems of managing this function. Most software changes are small from the user's point of view, but few changes are small from the maintenance point of view. Several organizational models have been tried for the management of support. None is entirely successful.

THE COMPUTING ENVIRONMENT

Through formal logging and recording of support requests and applied personnel time since mid-1976, Dow Corning's support group is able to report various characteristics of our applications support effort—about 60 person years of effort to date.

The following facts will give an idea of the environment and context of this support history:

- Each support person has been provided with a CRT and IBM's System Productivity Facility (SPF) timesharing software.
- Access to production source code is administered through a library control function.
- Batch turnaround time for compiles is usually less than 30 minutes, and probably 30% of the time it has been less than 15 minutes.
- Virtually all programming is done in PL/1, with small amounts of report programs written in Pansophic's EASYTRIEVE language.
- Administratively, requests are documented by the responsible key analyst as they are received (or encountered).
- The individual(s) working on requests logs his/her time on a daily basis but the time sheets are collected monthly for recording.
- About 50 individuals have had some part in this support activity over a period of $6\frac{1}{2}$ years. About a third was done by contract programming people.
- Table I gives some of the particulars as to the size and growth of Dow Corning's production environment.

THE MAINTENANCE LOAD

Some 60 person-years of data on support effort have been accrued over a $6\frac{1}{2}$ year period. About 64% was logged to individual requests; about 16% was logged against two general application support requests; about 20% was logged to nonproject time—e.g., vacation, holidays, and sick time. The following list of specifics relates to the 64% category of individual requests; it shows the important characteristics of the scheduling and performance of these requests.

Total number of logged support requests to date	4,454
Approximate number of open requests not completed (typically)	250
Total number of logged person-days to date	15,200
Total number of requests completed	3,690
Total number of requests canceled	460
Average size (in actual person-days) of completed requests. (See cumulative distribution curves, Figure 1)	2.6
Standard deviation of size of completed requests	6.4
Average turnover time (number of days from date request received until request was fully implemented and closed out) for all completed requests	73.4 days
Average "queue" time (number of days a request waited before being assigned to a support person to implement)	38.3 days
Average active time (number of elapsed days it took to complete a request once it was assigned to a support person)	35.1
Average age of current open request queue	354 days

Estimated backlog of work has ranged from 400 to 1300 person-days. Typically, the backlog estimate is near 600 person-days.

The average size of a request measured in effort to completion is 2.6 person-days; the average turnover time is 73.4

TABLE I—Growth of Dow Corning's production environment

Date	Production Jobs	Total Lines of JCL Code (thousands)	Active Production Programs	Total Lines of Program Code (thousands)	Average Lines of Code/Program	Production Runs per Month
1/1/73	647	37	N/A	256	N/A	2300
9/1/73	739	49	N/A	342	N/A	3230
9/1/76	1114	N/A	1863	440	236	5489
5/1/79	1444	164	2343	571	244	5523
9/1/80	1576	N/A	2693	N/A	N/A	6952
9/1/81	1662	218	2895	876	302	7202
9/1/82	1707	245	3282	1115	340	7793

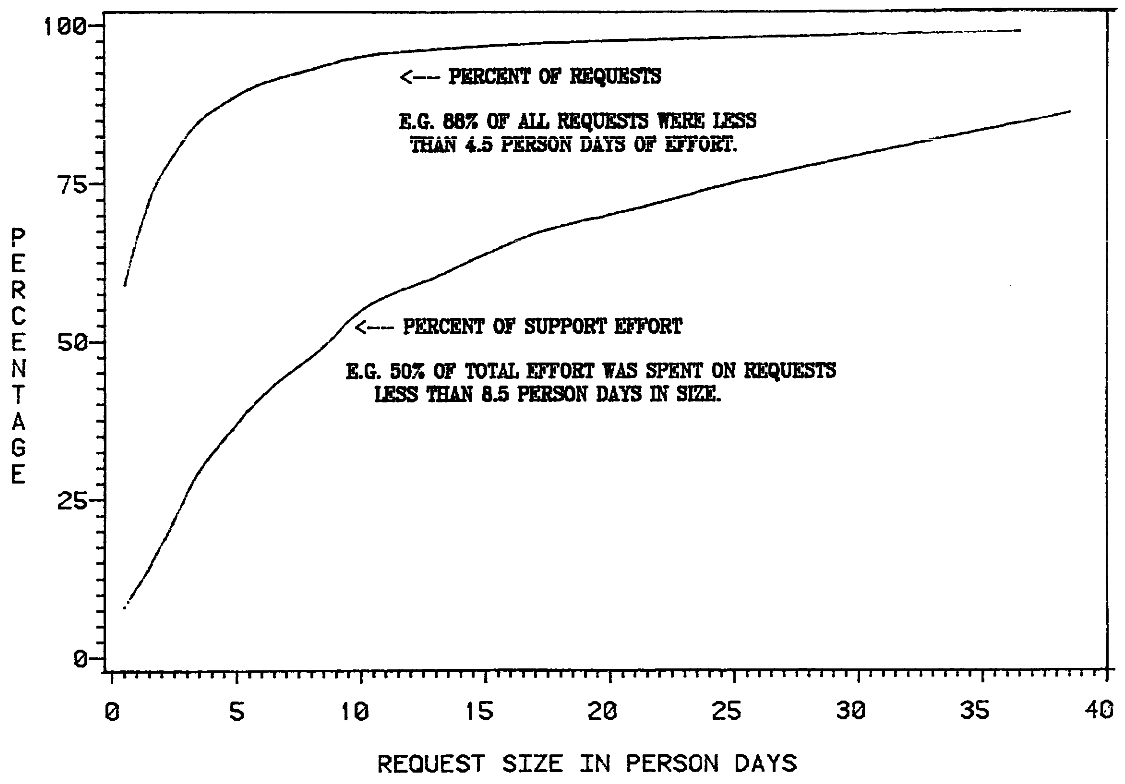


Figure 1—Cumulative distribution curves

days. Of the turnover time, about half is spent in the queue and the rest is spent while the request is being actively worked on. There seem to be valid reasons for this disparity between the size of the change and the duration of the change process.

First, changes in applications need to go through certain control and verification procedures before they can be considered as complete. This usually involves a designated *probationary* production period where the application is watched, and if unforeseen problems occur, the probationary time may be extended. Second, support people batch their effort by working on a number of requests concurrently.

Maintenance by Category

When requests are received or initiated, they are assessed as either *mandatory* or *discretionary* in nature. Mandatory requests require immediate attention—e.g., production application failures, executive edicts, and government requirements. Discretionary requests are all the rest, i.e., not mandatory.

Analysis of the most recent 19 person-years of support to distinguish mandatory versus discretionary effort is summarized in Table II. Individual comparisons of the last 2 years of the support group's efforts produced very comparable results showing a rather constant relationship.

The key analyst assigns a request type of code when requests are documented. The codes were established to categorize requests as to their primary reason for occurring, e.g., user request, consequence of some type of failure/problem, or preventive maintenance. Fourteen request type codes were used as follows:

User request types:

MI. Request for information about a system, e.g., how do I update this information?

MM. Maintenance due to management decisions, e.g., department reorganization, and new marketing strategy.

MP. Mass update of production files, e.g., change credit coding for a large group of customer types.

TABLE II—Mandatory versus discretionary support (19 person-years' experience)

	Approximate Number	% of Total
Mandatory requests supported	435	36.6
Discretionary requests supported	752	63.4
Totals	1187	100
Person-days expended on mandatory items	890	22.9
Person-days expended on discretionary requests	3002	77.1
Totals	3892	100

NJ. New job development, e.g., develop some new application.
RM. Request for discretionary modifications to existing production applications.
SA. Stand-alone request, e.g., write a program to do a one-time analysis of sales activity.

Consequences of some failure/problem:
MO. Maintenance due to operations' problems, e.g., ran job out of sequence.
MR. Maintenance in support of a restart, e.g., an application failed and the restart was not a minor experience. The failure is not directly attributable to any specific problem area.
MS. Maintenance due to systems design, e.g., a hard coded table ran out of space and caused a failure.
MT. Maintenance due to technology changes, e.g., new printer requires changes be made to some report generating programs.
MU. Maintenance due to user problems, e.g., user inadvertently input the same data twice.
MX. Maintenance due to user support function, e.g., inadequate testing of a change causing problems in the production environment.

Miscellaneous types:
MA. General maintenance request. Not attributable to any identified problem area, yet it is necessary in order to keep an existing application functioning—e.g., routine restart support, make small corrective changes to a program.
MB. Preventive maintenance—e.g., expand a field size before it causes a failure; correct edit logic oversight before erroneous data get passed into the system, etc.

Table III reports on the distribution of completed requests and the associated effort, the portion of support that was logged to specific requests. This relates to 10,200 person-days of effort, 64% of the total time recorded over some 6 years of experience. The average and standard deviation columns relate to completed requests only, leaving out canceled, withdrawn, and open requests. Completed request effort makes up 95% of the total effort reported.

TABLE III—Distribution of support by request type

Request Type Code	% of Time	% of Requests	Average Size (Person-Days)	Standard Deviation
User Types				
1. MI	4.4	9.8	1.1	2.9
2. MM	5.3	4.0	3.4	6.8
3. MP	.2	.4	1.4	2.2
4. NJ	18.7	4.3	14.9	17.0
5. RM	23.8	16.8	4.5	7.2
6. SA	14.4	8.1	4.8	10.2
TOTALS	66.9	43.4	4.4	9.0
Failure Types				
7. MO	1.9	5.6	.8	1.2
8. MR	2.0	3.2	1.6	3.7
9. MS	6.0	8.8	1.7	4.4
10. MT	1.7	3.8	1.1	1.6
11. MU	4.9	9.8	1.2	2.8
12. MX	3.5	4.6	1.9	3.9
TOTALS	20.0	35.8	1.4	3.3
Miscellaneous Types				
13. MA	12.4	18.6	1.7	3.9
14. MB	.7	2.2	.9	1.0
TOTALS	13.1	20.8	1.6	3.7
FINAL TOTALS	100%	100%	2.6	6.4

ORGANIZATIONS AND APPROACHES

Like many healthy, growing companies, Dow Corning over the past decade has seen fit to reorganize its management information service (MIS) function many times. Each new or modified structure had its own rationale and was an attempt to improve the effectiveness of the MIS function. The organizations that reflect the way maintenance effort was addressed or managed will be described.

About 11 years ago, the MIS function of Dow Corning had reached the point of a fairly respectable shop:

1. Approximately 500 production batch applications with online order entry and customer file maintenance
2. IBM 360/50 computer; disk and tape drives
3. Operating three shifts per day
4. About 14 people providing technical programming and system design skills
5. One of IBM's first PL/1 shops

MIS Organization: 1971–1974

Organizationally, (see Figure 2) MIS included within its domain three user representative positions entitled Functional System Supervisor. The staff had dotted line connection with the functional area they were associated with and with responsibility for defining and specifying the user requirements for almost all application support requests, maintenance or otherwise. The maintenance programming group consisted of one or two former computer operators who were learning JCL and PL/1 programming on the job. Using Lientz and Swanson's[1] definition of maintenance, which "refers to all modifications made to any existing application system, including enhancements and extensions," about 50 to 75% of all programming effort was maintenance. Perhaps a majority of this effort would better be described as incremental extensions of base systems, as opposed to corrective or other types of maintenance.

The programming and analysis group consisted of about six or seven technical programming people who received requirement specifications from the functional system supervisor af-

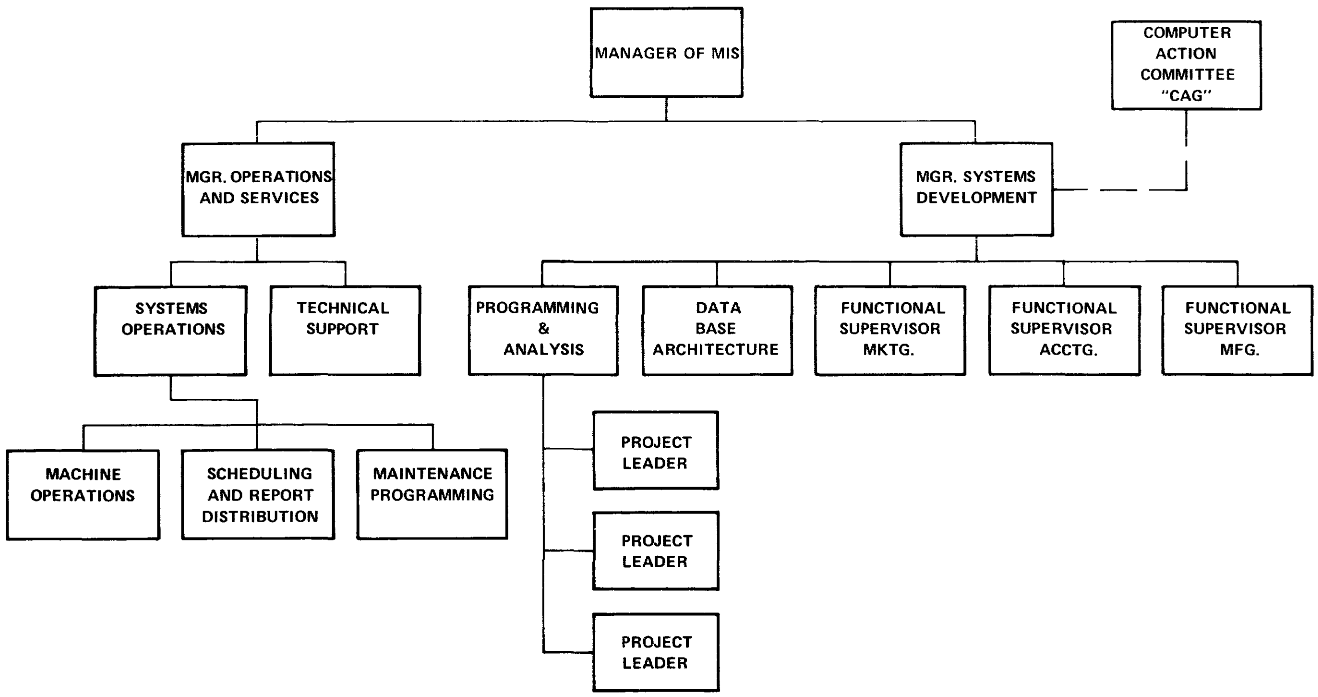


Figure 2—Basic MIS organization 1971–1974

ter they were first approved by the supervisor of the programming and analysis group. Because the functional system supervisor was intended to be occupied with longer term and larger application activity, users in need of more immediate support of small projects were considered a detraction to the functional system supervisor. To provide for this, a small projects group was set up with a leader and a variable resource budget equivalent to a staff of one or two. This budget was reconsidered on a quarterly basis, by the Computer Action Committee. This committee consisted of a group of key user representatives from the main functional organizations of Dow Corning who approved the project plans and resource schedule proposed by the systems development manager. This basic organization continued until the time frame of the oil embargo and the corresponding economic slowdown of 1974.

MIS Organization: 1974–1976

As a result of the slowdown, large project activity was curtailed, the functional system supervisor positions were eliminated and the systems development manager position was vacated and an expanded small projects group was renamed User Support Group (see Figure 3 for this organization struc-

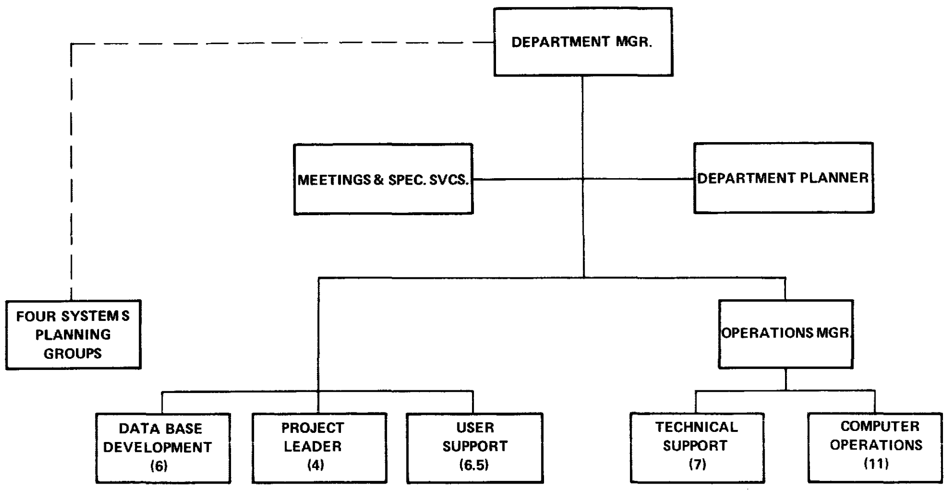


Figure 3—Basic MIS organization 1974–1976

ture). Coupled with the establishment of this new group was the intent that all small projects activity (including operational maintenance support) would be done exclusively by this group of about six full-time and one part-time person. This represented about 40% of the application programming resources in the department at that time. A 30 person-day maximum was defined as the threshold for classifying a request as a small project.

The title, User Support Group, was picked to emphasize the *user* interface role of this group. This role included some technical consulting to aid users in learning their own systems, as well as learning elementary DP skills for creating their own reports from production files.

The user interface with the User Support Group was with the four chairmen of the functionally aligned planning groups. Requests for small projects were received by the User Support Group supervisor and he made time estimates to accomplish the tasks. There was always a backlog of work requests, and on a quarterly basis, the User Support Group supervisor met with the Systems Planning Group representatives to communicate progress and decide on priorities for all the open requests. This basic structure continued until late 1976.

Small Projects Separated From Maintenance: 1977

Better economic conditions allowed for a revitalization of the Systems Development Group, but the User Support Group was moved under the operations manager position (see Figure 4). Part of the reason for this move, was to enable systems development to concentrate on new systems exclusively.

Another attempt was made to have the USG staff of eight responsible for small projects only, and a separate group of four responsible for maintenance, all under the operations manager. At this time, Systems Development had a staffing level of fifteen. Maintenance was defined as *any work (on a pre-existing system) necessary to keep the system functioning as it was intended at implementation.* Such maintenance requests tended to average about one and one half person-days effort, whereas small project requests averaged about six person-days.

The separation of a maintenance group from the small projects group came to be viewed as being inefficient. Both groups had a supervisor who separately communicated with the functional representatives now called functional system coordinators. The nature of the communications was the same, i.e., setting priorities and communicating progress or problems. Furthermore, the nature of the work was the same and there was confusion among all parties as to what category a particular request might fall under. Was it maintenance or was it a small project? A larger number of requests were addressed under this approach, compared to the years before and after. This may have been because more resources were concentrated on this category of support maintenance that typically required less effort per request to complete than the small project request.

During this period an MIS departmental procedure was formally established that required User Support Group approval of new applications before they were considered to be in a fully supported production status. Our shop started being more sensitized to the desirability of formally documented standards for production applications.

One of the benefits of having the maintenance function under the operations side of the MIS organization was the degree of checks and balances that tended to consider the long term operations concerns of an application: Was the job conveniently restartable? Were there backup provisions identified for permanently created files? Are the reports uniquely identifiable to enable their routine distribution? Are error interrupts appropriately reported? Does the job delete all its temporary files upon completion?

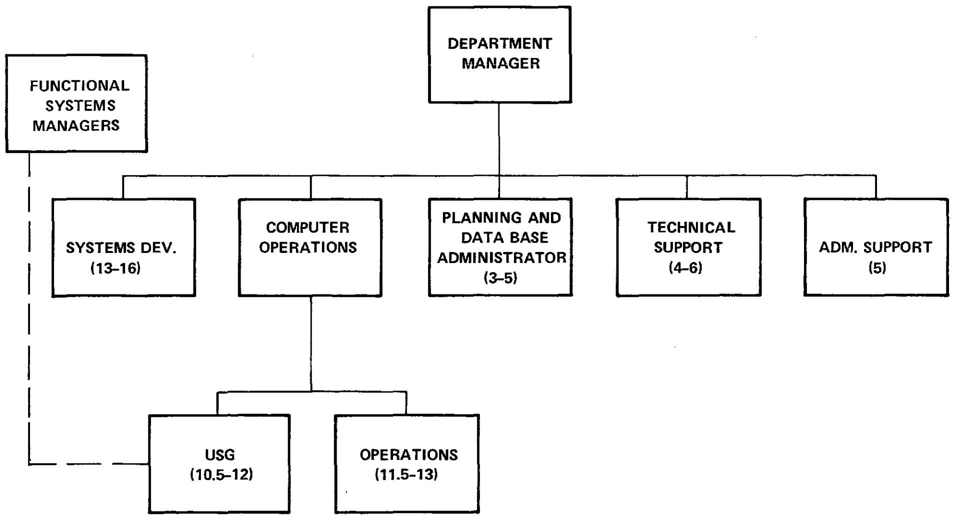


Figure 4—Basic MIS organization 1977 and onward

Small Projects and Maintenance Reconsolidated: 1978

A divided maintenance group was reconsolidated and the work activity was viewed as being either mandatory or discretionary after a year. The mandatory work activity would be addressed immediately in order to maintain or restore integrity to the production applications, while discretionary requests would be prioritized by the user organization. The previous practice of meeting every couple of months with the systems coordinators was continued. A joint consensus was reached at these sessions as to what was most important to the *corporation,* rather than deciding according to a strict *functional* perspective. In practice, this caused dissatisfaction among some of the systems coordinators who could not plan on any particular program because their requests were superseded by new, higher-priority needs in other functions. Some coordinators thought that their functions had had, and continued to have legitimate needs, but could not compete favorably with the more visible and always-pressing needs of other functions.

Current Approaches

This led to the approach that Dow Corning has used for the last four years. Each function is allocated a percentage allocation of available user support resources. This percentage is renegotiated on an annual basis. In addition, the functional systems managers (as the functional systems coordinators were renamed) could, on an individual basis, provide funds to augment their allocation with contract programming resources or they could negotiate with their functional counterparts to trade resource time or temporarily acquire a larger percentage allocation. It was expected that this would enable the functional systems managers to better plan their own destiny and eliminate some of the haggling of group priority-setting sessions. This was largely accomplished, but not without some other effects.

In 1977, Dow Corning management developed and formalized the current organizational structure on the user side. The Systems Management Board is made up of top level executives from Dow Corning's key operating functions. Subordinate to each of these executives is a functional systems manager. The Systems Management Board sets the broad direction for systems at Dow Corning. The functional systems manager coordinates the development and operation of information systems for his function and is the primary liaison with all MIS department managers.[2]

Each functional systems manager was charged the time expended on mandatory maintenance for their applications as they were encountered. This was somewhat predictable, based on historical averages of 20–25% of total support allocation. The annual exercise to establish new percentage allocations for the new year met with disagreement and the easy way out simply perpetuated the status quo. The interfunctional trading or surrendering of resource time has occurred only to a limited degree. This has resulted in an effective reduction of the corporate perspective in our systems maintenance. Some functional areas have what appear to be important needs waiting while other functions might simply be using their time to provide only marginal benefits to the corporation.

The User Support Group organized its activities along functional lines and the User Support Group name was exchanged for Production Systems Support. The word *production* was used to reflect the exclusive production emphasis of this group of resources. Support of documented and accepted production applications or support of new applications being developed for documented production status could normally be considered valid activity for Production Systems Support. (The 30 person-day limit was still a limiting constraint for all requests or groups of related requests.) This meant that Production Systems Support would not provide consulting or programming services for individual users unless this criterion was met and the activity had been given priority approval by the associated functional system manager. There had not been that much non-production activity going on but with more and more user computing anticipated, it was recognized as a difficult activity to support and control centrally and still keep up with the demands for support of production applications.

The Production Systems Support Group organization (see Figure 5) identified a lead analyst role entitled key analyst. Each key analyst has primary responsibility to oversee all Production Systems Support efforts that affect his/her functions' applications, about 425 batch jobs and 2 online systems per analyst. The functional systems manager is the key analyst's user contact for purposes of receiving change requests and receiving priority assignments for requests. Mandatory priority status can be assigned by the key analyst or the systems manager.

REMAINING PROBLEMS AND POTENTIAL SOLUTIONS

Development of Production Systems Support staff in the latest technology is difficult and impractical. Ever present user demands and mandatory maintenance produce an environment that will quickly develop staff in the traditional DP technology used in existing systems, but it does not lend itself to learning the latest technology being implemented in the new systems. The movement of people between development and applications support can alleviate this, but it does not naturally happen, especially when groups are under different managers.

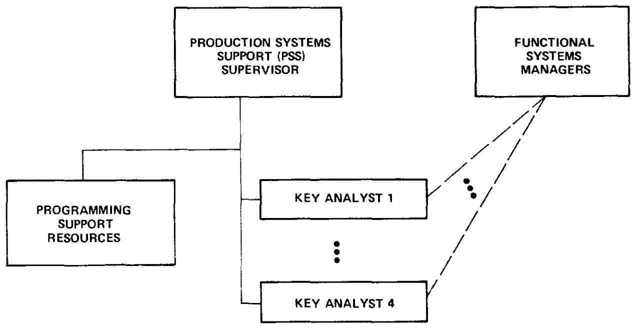

PRODUCTION SYSTEMS SUPPORT (PSS) SUPERVISOR
FUNCTIONAL SYSTEMS MANAGERS
PROGRAMMING SUPPORT RESOURCES
KEY ANALYST 1
KEY ANALYST 4

Figure 5—Organization of the PSS group

Personnel motivation and enthusiasm can be difficult under normal circumstances, but problems are compounded when the staff is given low priority work in the less active or overbudgeted functional areas.

The intermediate sized project (30 to 100 person-days) is very awkward to handle. Such a project is too large for the support groups' resource level and too small to disturb the progress or plans of the systems development group who are usually involved in much larger projects. The problem of lengthy elapsed times from assignment to completion of even the small requests remains an undesirable characteristic of our experience.

Consideration is now being given to a couple of changes that might serve to solve some of the past problems without introducing many new ones. One change would consolidate most of the Production Systems Support resources with the development resources, but retain the existing functional key analyst position of the support group. A small portion of the support group would stay in the operations structure to provide basic operational maintenance support for failed applications. This change would help resolve the technological development problem of support staff since rotation of support staff would be less difficult to arrange under the same manager.

The second change deals with the manner in which resources are allocated to the functions. Rather than each function having a percentage of the total support budget so that all the percentages sum to 100, a portion of the support budget would be allocated. This portion would be determined by what was considered slightly more than sufficient to handle all mandatory maintenance. The remaining portion would be applied to support Production Systems Support projects as decided by periodic priority-setting meetings of the functional systems managers. This change would make it more practical to handle the intermediate size project, since if desirable, all the discretionary resources could be applied to the task so that it could be completed in a reasonable time frame. Furthermore, all the functions would get some minimal level of resource, but a good portion of the effort (about half) would be more consistently applied to the highest-priority corporate need. Another alternative to settling the resource allocation questions would be to charge back time directly to the functional departments, thus letting the buyer decide his/her budget.

One potential weakness seen in consolidating the support resource with systems development is the organizational loss of commitment to operational concerns. Perhaps a strong operations representative on project review committees would alleviate this weakness. Another area that could introduce frustration and inefficiency is overlap of responsibilities. The operations support activity would have to be controlled and adequately distinguished from the production systems support activity. Perhaps making the operations role one of a short-term perspective (to keep applications running on schedule) would enable an adequate delineation of responsibilities to be made.

Chargeback of time could introduce difficulty in maintaining a stable staffing level—low budget years would probably translate into dramatic reductions in support purchases.

WHAT IS THE ANSWER?

Managing application support effort is fraught with dilemmas:

1. A large demand for changes coexist with demands for new systems—how is an effective balance defined and achieved?
2. Personnel development in new technology is not practical when support people are fed a diet of traditional DP demands.
3. How can support effort be distributed to effectively meet corporate needs while retaining viability with individual corporate functions?
4. Where do you put support activity and still retain operational interests along with personnel development needs?
5. How do shops reduce the "active time" of a request and still retain needed verification and control features?

Different DP organizations share these basic problems, although the size of the shop will alter the extent to which some of these areas actually cause serious concern. Are there any management models that have been developed and tested that address this business situation? Has a consensus of opinion or experience been identified?

REFERENCES

1. Lientz, B. P., and E. B. Swanson. *Software Maintenance Management.* Reading, Mass.: Addison-Wesley, 1980.
2. Closs, J. P., and J. E. Randall. "Dow Corning Improves Group Communications." *INFOSYSTEMS,* August 1980, pp. 70–72.

Organizational issues of effective maintenance management

by GARY L. RICHARDSON
Texaco Inc.
Houston, Texas
and
CHARLES W. BUTLER
University of Arkansas
Fayetteville, Arkansas

ABSTRACT

It is a continuing challenge to today's data processing (DP) organization to evolve a management structure that matches the technological advances of the system for which it is responsible. The goal of this paper is to synthesize the emerging role of a DP organization within its corporate environment, focusing particular attention on the issue of software maintenance.

Attention is primarily on three dimensions of the required organization: the user view, organized by functional area; the technical view, organized by area of expertise; and the organizational view, arranged by planning horizon. The conclusion is drawn that a single group should have responsibility for integration and enhancement of all installed applications. The tasks of this group comprise software configuration control, operational integrity, performance tuning, and requirements analysis and planning support for installed systems.

INTRODUCTION

Today's data processing (DP) technology is headed in many different directions at once. This explosion in technology, together with twenty years of systems development, has suddenly brought us to the exciting threshold of new organizational strategies and direction. The goal of this paper is to present a generalized organization structure that facilitates development and enhancement of software, focusing particular attention on the software maintenance function. The ideas raised here are meant to be independent of any single organization and are designed to help the reader sort through the myriad of factors that affect a particular organization. However, it should be emphasized that the approach presented has been applied within a large DP organization.

THE EMERGING DP ENVIRONMENT

A practical framework for analyzing the DP function can be formulated by reviewing critical operational components and observing their unique pattern of evolution or growth. In the final analysis, organizational structure depends on three distinctive factors: data resources, technological components, and the application environment. Because of the integration of various levels of technology, the organization generally evolves according to the pattern outlined in Table I. The driving force in these evolutionary stages is hardware technology, which supports increasing capability, as demonstrated by today's complex network configurations and operating systems. This fact is illustrated in Table I by the diagonal arrows, which indicate the pushing effect of technology on the application environment and organizational framework. When one compares it to these technological advances, it is apparent that the evolution of application software and organizational framework has lagged behind technical sophistication.

TABLE I—DP Components and Organizational Trend

	Factors			
Observation Point	Data Resource	Technological Component	Application Environment	Organization Framework Required
I	Records	Mono-Programming	Application	Narrow *(I/O)*
II	Files	Multi-Programming	System	Expanded *(i/p/o)*
III	Database	Virtual Machines	Multisystem Integration	Functional Specialization *(I/P/O)*
IV	Information Center	Networks	Virtual Application	Multidimensional Support

Organizational growth is continuous, therefore difficult to describe in discrete intervals, but recognition of four points along the continuum provides critical insight into the emerging organization. The four points are as follows:

1. Narrow.—Low personnel requirements, data maintained as a separate file for each application; focus is on input and output (I/O) and on replacing existing manual procedures.
2. Expanded.—Extension of application functions or processes *(i/p/o)* and an accompanying growth of personnel resource requirements. Processing becomes more complex.
3. Functional specialization.—Integrated systems composed of major functions and processes *(I/P/O);* personnel with specialized knowledge within critical functional areas.
4. Multidimensional support.—Automated applications cover the firm's entire operating sphere; information centers emerge that focus on query and decision-oriented areas.

In the past, applications such as payroll or accounts payable were designed with limited scope. Today, each of these is merely a subsystem, a single element of multisystem integration. The firm is now operating these *virtual applications.* They are critical in that they are embedded in the firm's functional operating environment. However, even with this level of sophistication these systems are never quite finished, owing to the dynamic nature of business operations.

Along with the existing virtual system complexity, most large organizations have also been expanding their services in multiple directions. A sample of these is as follows:

1. Providing on-line access to users
2. Installing nonprocedural software for users
3. Directionally moving toward the information center concept
4. Decentralizing installation of personal computers
5. Installing text processing and/or electronic mail systems
6. Connecting multiple processing and user nodes into distributed networks

It takes only a casual glance at this list to see that the scope of work undertaken has broadened considerably. It is not so obvious what pressure this brings on the traditional DP organizational structure and on its procedures for maintaining existing software systems. The breadth of hardware and utility software used is too much for one group to handle. In other words, the new environment demands a different organizational approach to meet the breadth and complexity of demands being placed on it.

DESIGN FACTORS

Operating Factors

The fundamental issues affecting the structure of organizational units should be examined in such a way that the following five questions will be addressed:

1. What is the perceived goal of the information systems department?
2. Should software systems be carefully planned before coding, or is prototyping the best technique?
3. Should development work units be organized primarily around skills or around user functions?
4. Could your organization operate adequately if the computer system were lost through a catastrophe?
5. What are the typical problems encountered by users?

Obviously, these questions can be answered in a wide variety of ways. Collectively they indicate the continued need for an application development unit.

First, the goal of the information systems department is more and more recognized to be the maximization of the value of corporate information (including data, graphics, video, and voice, as well as text). The chore of writing new traditional systems code is declining relative to the new support services such as telecommunications (networks), training users, decentralizing hardware, and managing the orderly growth of computer technology within the total organization. This suggests that user involvement with the organization is broadening in scope beyond the functional systems specification activity.

Second, if we can prove that the use of good software engineering practices does not adversely affect delivery times for new systems or enhancements, there will be increased justification for splitting system specification and analysis away from code construction. However, a large segment of DP professionals believes that prototyping is the only way to find out what users really want. Quite probably there will always have to be some prototyping in the process of collecting specifications; possibly skill in prototyping is needed in applications development.

Third, development work needs to be controlled by professionals who understand both the user requirements and enough about code construction to make intelligent design tradeoffs. We propose that development groups be supplied with carefully described specifications, be predominantly skill oriented, and work from formal design blueprints, just as carpenters would in building a house.

Fourth, if you feel that the computer is not indispensable, then your organization has not reached the evolutionary stage we have been describing.

Fifth, from the list of user services developed from question 1 you should observe a wide assortment of requests and problems, embracing such items as data availability, purchased software, micros, remote terminals, local training, and so on. The list should definitely include more than just simple application development system software concerns. The truth of this premise would mean that we really cannot simply organize around user requirements. They would need the entire organization! Rather, we must consider techniques to provide a simplified organizational view to users while hiding the true organizational complexity away—in other words, a multidimensional structure.

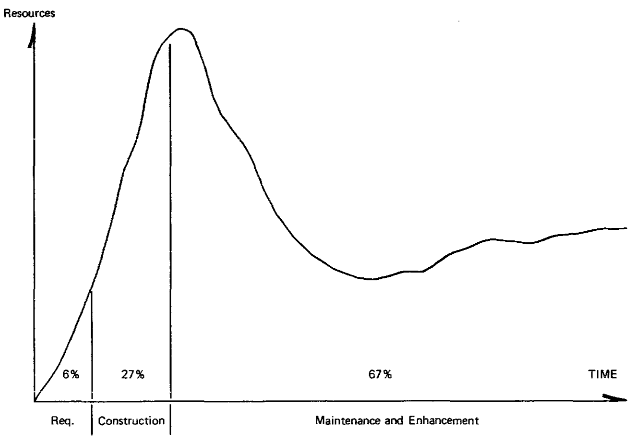
Human Factors

Human factors are also important in the design of a DP organization. It is important to consider, for example, whether computer professionals naturally relate to users or to technology. Research evidence indicates that they are primarily technology oriented and that well-trained individuals can learn how to translate a reasonably well-defined user problem into a computerized system. This suggests that staffing is easier when it is based on skill groups, rather than on functional applications. If requisite skills within the organization were homogeneous this would be a moot point. That is not the case today, and the cost of broadly training individuals is becoming quite noticeable. In addition, there is a limit to the range of diverse technology an individual can be technically proficient in. A natural conclusion is that skill-centered units are the most natural organizations for the application development and technical-support functions.

Organizations can hinder or help human productivity. In order for a professional to understand his or her role, the organization must provide a coherent structure for logically perceived work processes based on a division of labor. One more decade of developing and operating information systems with current techniques and philosophy will bring even greater chaos. Too many options are being generated by technological development, and management is increasingly looking at computer technology as a tool, rather than a status symbol. The organization cannot tinker with technology and human resources; it must apply good business decision-making principles.

THE IMPORTANCE OF SOFTWARE MAINTENANCE

Is maintenance important? The answer to this question is unequivocably yes. We are now entering an era in which the operational nature of most software systems is so intermeshed with functional systems that large segments of the organization cannot perform their duties when the computer system is not operational. Changes to these systems are increasingly complex because of this and because of the integrated nature of the systems. Finally the investment that firms have in installed systems is becoming widely recognized, as system retrofit activities highlight their replacement cost. In order to extend system life, the role of maintenance is critical; it is clearly significant for the system life cycle. As illustrated in Figure 1, the system life cycle can be viewed as a series of



Source: Zelkwitz, M.V. "Perspectives on Software Engineering," ACM Computing Surveys, Vol. 10, No. 2, 1978, pp 197-216.

Figure 1—System life cycle

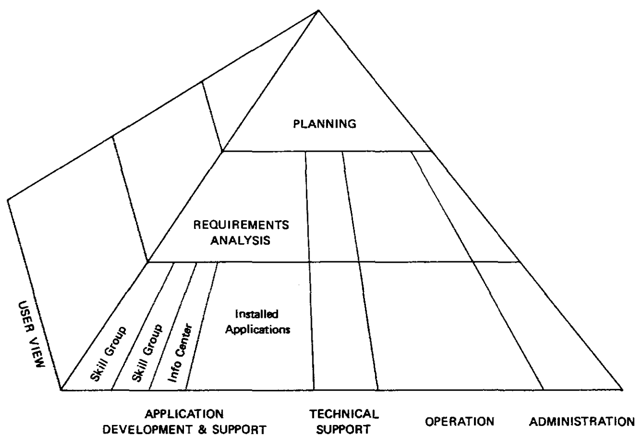


Figure 2—Multidimensional organization structure

broad subprocesses: requirements definition, construction, and maintenance/enhancement.

Another critical factor that helps to explain the importance of maintenance is the scope or impact of maintenance changes. Research by Swanson[1] and Boehm[2] aids in sorting out the importance of change scope. They divide maintenance work into four levels of change scope: (1) corrective, (2) adaptive, (3) perfective, and (4) expansive. Comprehensive understanding of existing software is required for successful maintenance changes. Corrective and adaptive maintenance requires an understanding of existing software and is generally restricted to small segments of an existing system. As the scope of a change progresses into the perfective and expansive categories, system performance and functional enhancement become the primary goal. In these instances, more knowledge of component interaction is needed, even though relatively small amounts of code are used for successful changes. In essence, reevaluation of system construction is performed.

Three conclusions can be drawn regarding the nature of software maintenance. First, the demand on the maintenance function depends on the quality of products that are generated during the development phase of the life cycle. Second, regardless of the quality of the software product, maintenance success depends on a level of understanding of the existing system. Finally, even the smallest software change is significant because of the value of software as an asset.

GENERALIZED ORGANIZATIONAL STRUCTURE

A multidimensional organizational structure is required to assimilate technological innovation properly, address broad user needs, and manage the software life cycle properly. Each of these requirements is addressed in the organizational structure illustrated in Figure 2. First, the horizontal axis represents the technical scope of the DP organization. By organizing along technical skill lines, specific skill requirements can be developed. Next, the vertical axis represents an appropriate packaging of the key functions of the DP organization. The various work units do not extend continually up the organization, because at higher levels less technical and more managerial skill is required. Finally, the third dimension represents the extension of data processing into the firm's operation and is a functional user view of the organization.

Given the wide range of technological alternatives and diversity of user requirements, at least five major organizational units are required for multidimensional support. As shown in Figure 2, they are as follows:

1. Planning—Development of strategic and tactical plans, including (a) the primary planning interface to the management of operational departments and (b) evaluation of technical advances.
2. Application development and support—Software development and support activities that are taken within the department.
3. Technical support—Technical support functions required for the department to perform its responsibilities (systems programming, DBA, telecommunications, etc.).
4. Administration.—Administrative functions involving the personnel and financial activities within the departments' operation.
5. Operational support.—Operation of the computer centers and remote I/O stations.

Salient to the issue of effective life cycle and maintenance management is the applications development and support unit. Our premise, which is supported by research conducted by Mooney[3] and Canning,[4] is that applications development and support will be composed of two distinct responsibilites: development and installed applications. Figure 3 is an example of application development with four skill environments (e.g., COBOL, PL/I, APL, and micros). The skill requirements shown are intended to be development-oriented skill groupings. For instance, a COBOL group would be distinct from a PL/I group. The point is that skills are rarely intermingled within the same development work units.

Note that the traditional maintenance function is retitled

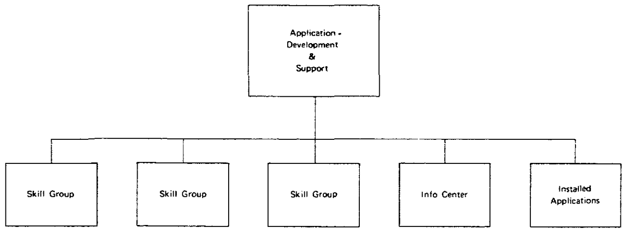


Figure 3—Application development overview

"Installed Applications" and made responsible for the proper functioning of all installed systems, as well as for minor enhancements. It focuses on software configuration control, operational integrity, performance tuning, and requirements analysis and planning support. It should be staffed with an appropriate mixture of professionals and senior personnel who can handle the broad range of tasks implicit in this function. The installed applications will require managerial and technical skills critical to the activities of software maintenance.

As a responsibility of the application development and support unit, maintenance activity typically consumes more than half of the development function, and the growth of virtual applications will bring additional support requirements into the maintenance programming sector. In this environment, it is necessary to learn how to efficiently maintain installed systems to provide optimal resource allocations in new development areas and a stable data resource base for other applications to use in a decision support mode. The installed applications group concentrates on operational systems, which are a basis for shared data in the firm. The proposed structure offers the operational benefits of a wide range of technical skills within which maintenance is a significant component.

MAINTENANCE MANAGEMENT BENEFITS

As stated earlier, an organizational structure can only facilitate productivity. It is up to management to optimize human and technological resources. Operating within the general organizational structure, the installed applications unit presents a number of operational and managerial features for maintenance management:

1. Quality assurance.—Experience reveals that the requirements definition (analysis) is heavily user dependent, while the construction (design, coding, and implementation) is more heavily slanted toward the hardware/software environment. After implementation, the operation/maintenance phase requires both user knowledge and technical skills at various times. In all circumstances, maintenance is performed on products originally developed through requirements definition and construction. A final system acceptance point is established between the two subprocesses; inferior products can be rejected at that point.
2. Low prestige factor.—Separation of functions can enhance image through top-level management recognition and participation. When a development project is completed it is transferred into the installed applications unit along with a skeleton support staff.
3. Resource level determination.—Resource allocations to a recognized function should result in equitable human resource allocation for maintaining the valuable software asset.
4. Unique skill requirements.—Educational and training programs can be established for the function, not the individual. Special tools can be developed or acquired to complement the function.
5. Professional development.—Job rotation is handled at the planning level, aggregating individual needs and training objectives. Thus, a professional can perceive that he or she is not buried in the installed applications area.
6. Change integrity.—Through orderly job rotation a constant base of knowledge is maintained. By organizing maintenance as a single function, better internal control can be implemented to ensure that changes are authorized.

CONCLUSIONS

From a practical point of view, installed applications are an unavoidable fact of life. It has been stated that "the main problem in the maintenance business is that you cannot just do maintenance on a system which wasn't designed for maintenance. Unless we design for maintenance, we'll always be in a lot of trouble after a system goes into production."[5] Recognition of this has legitimized many of the methodology developments during the past decade. Unfortunately, it glosses over the major problems and fails to address yesterday's legacy of a voluminous software inventory that we are maintaining today.

The primary objective of this paper was to focus on organizational factors that facilitate maintenance management. However, the complexity of the function and the related sensitivity of the organization to the subject required that this discussion go beyond the single issue of maintenance. In particular, there are three major elements that must be integrated to effectively provide support for installed applications; they have been identified as follows:

1. Organizational structure.—a formal organization architecture within which resources are distinctively responsible for maintaining the firm's applications.
2. Clearly defined work process.—an identified life cycle within which the natural growth of installed applications can be managed and controlled.
3. Special technology—unique manual and automated aids which serve as diagnostic, investigative, and corrective mechanisms for today's complex applications.

Within each of these areas, managerial expertise is required to overcome inherent problems of morale, professional development and training, coordination between work stages, assimilation of technical tools, and integration of multi-dimensional support. It is hoped that the reader can identify pertinent factors that apply to a local organization and use them in constructing a path for his own organization.

Many well-read writers such as Martin and McCracken declare that the coming of fourth generation software is going to allow users "simply" to query mystical databases and answer whatever question is on their minds. This sounds neat, but our experience with approximately 1,000 such users suggests that user training, consulting, database design, and data migration are all left behind with the computer professional. While we hope that this approach will slow the growth of expensive maintenance for traditional software and cut user backlogs, we have recognized that maintaining corporate software assets is a critical DP function.

ACKNOWLEDGMENTS

The authors wish to acknowledge the contributions made by L. J. Crockett and D. W. Hutzelman of Texaco's Computer and Information Systems Department in Houston. Much of the structuring concepts presented here resulted from discussions with them; their input helped shape many of the ideas expressed here.

REFERENCES

1. Swanson, E.B. "The Dimensions of Maintenance." *Proceedings of the IEEE/FCM 2nd International Conference on Software Engineering,* October 1976, pp. 492–497.
2. Boehm, B.W. "Software Engineering." *IEEE Transactions on Computers.* C-25 (1976), pp. 1226–1241.
3. Mooney, J.W. "Organization Program Maintenance". *Datamation.* 21 (February 1975), pp. 63–66.
4. Canning, R.G. (ed.) "That Maintenance Iceberg." *EDP Analyzer* 10 (1972).
5. Lientz, B.P., and E.B. Swanson. *Software Maintenance Management.* Reading, Mass.: Addison-Wesley, 1980.

When a data processing department inherits software

by JOAN R. ZAK
Henry S. Miller Companies
Dallas, Texas

ABSTRACT

This paper discusses some of the problems that occur in dealing with inherited software and some of the basic procedures necessary to manage successfully a data processing department or group that is converting and/or maintaining software that has been imposed on them and that they neither designed nor implemented. A significant constraint is the impossibility of contacting the author(s) for guidance and help.

The two situations described are very different, yet they share some problems. The first involves an engineering company with a relatively scientific, real-time application—the design of a security system for a nuclear power plant. The second is a real estate management company with fairly typical business applications.

This paper does not hope to provide all the answers, but instead to help raise some of the questions that arise when dealing with inherited software.

INTRODUCTION

Every maintenance programmer is faced with the problems of inherited software—trying to fix or improve a program that someone else wrote. However, a whole new level of complexity is introduced when the entire shop is working with software they neither wrote nor designed. In this case there is no one to go to who understands what the system does, no one who might have a clue about what the author was thinking when the program was written. The programmers can have difficulty in understanding some of the broader background issues involved at the time the system was written—issues which may or may not still be applicable.[1] The process is painful, and, if not well managed, can cost the company a lot of time and money.

I have been in two situations where an entire group or data processing department has been working with inherited software; and I would like to share pitfalls, special problems, and my recommendations for working in this kind of environment.

SITUATION 1—THE DISASTER

Background Information

The first situation was an engineering company in Texas, whose sister company had won a bid to build a security system for a nuclear power plant. The sister company folded after working on the project for a year and collecting about 90% of the money. The programmers assured the people who were coming to take over the project that the system was almost complete. In fact, they grumbled that if they had just been given another month or two, the entire system would have been working.

The security system was supposed to run on a PDP-11/34 with 256K bytes of memory and 15 megabytes of disk space. The system was badly underconfigured, since over 2,000 detection and alarm devices were supposed to be connected to it. The relations with the customer were very negative. The customer had been burned once by the other company and was going to make sure it didn't happen again. Therefore, the customer was not open to suggestions about buying more hardware.

The engineering company that took over the project had a big problem with lack of expertise, especially in software. They hired, as programming manager, a woman who claimed she had 7 years' experience on a similar operating system, but who turned out to have very little knowledge about this system. They hired a number of consultants to work on the project, all claiming expertise in one part of the system or another. By the end of the project two company employees, nine consultants full time, a representative from the customer's company, and a Florida consulting firm were involved.

The Scope of the Project

Documentation consisted of the 300-page requirements document prepared for the customer by another consulting firm, a 300-page functional specification, and a system flow chart. The requirements document asked for things that were almost impossible to do, the functional specification was vague and contradictory, and the system flow chart was incomprehensible.

Of course, there were the programs themselves, consisting basically of undocumented FORTRAN and assembly language code.

Initial Attempts

The primary directive for the first 9 months of the project was to use as much of the existing software as possible. One of the employees started the job with about 9 months' programming experience, then spent the first 6 months of her job reading the programs and trying to understand what they were trying to do. Another consultant spent 3 months trying to modify the existing alarms programs so they would work. He was unsuccessful. My solution was to look at the code for about 10 minutes, announce that it wasn't worth saving, and begin to write the program from scratch. Unfortunately, I was one of the more productive members of the group.

In the long run just about every piece of software that the bestowing group had written was thrown out and completely rewritten. In some cases, although the program would have worked all right, it had to be discarded because it had been written in FORTRAN and was therefore too big to fit on the system.

The system was finally delivered—8 months late, and not meeting all the performance criteria.

Effects of the Decision to Use Existing Software

The most devastating effect of the decision to try to live with the existing software was that no meaningful attempt was made to do system design. All design decisions made by the first group were kept, which meant that no one had a real, thorough understanding of how the individual pieces of the system composed an integrated whole. They also did not have the proper tools to make an adequate analysis of the system. In particular, the three basic maintenance productivity aids—tools, techniques and training[2]—were almost totally ignored. For an asynchronous system, those involved were using synchronous tools, the system flow chart and the functional specification.

This lack of coordination was felt in other areas. Although at one point there were 15 programmers working on the system (most of whom had not known each other, much less worked together before this project), there were no coding standards, and no teams were formed. This is somewhat surprising, considering that Lientz has found that almost 80% of the shops he surveyed used at least chief programmer teams.[3] All programming was done on an individual basis, with no walkthroughs or group checking. One attempt by the programmers to hold weekly status meetings met with management disapproval; and although the meetings were extremely useful when they were held, they finally disintegrated. Thus the system was coded in 15 different styles, which is going to be a real maintenance headache.[4]

What Should Have Been Done

Now that you have a picture of what the project development process was like, let me explore with you what should have been done. I left the project before it was finished, so I don't know if a post-mortem evaluation was done, but I doubt it.

Systems analysis

First, the company should have evaluated the system requirements by using an analysis tool that could give them an overview of the entire system, such as Data Flow[5, 6] or Warnier-Orr[7] diagrams. This would have helped the managers and the participants to see what the system was doing. As it was, only one or two people (if any) had a reasonable idea of how the system fit together, which meant that a great deal of redundant and counterproductive code was generated. In a way this recommendation is rather silly, because none of the people on the project knew that these kinds of tools existed—or, if they did, wouldn't admit it.

Evaluate system requirements

The second step that should have been taken is to evaluate the limits of the system's requirements. The project started in October, and the following September I finally did a memory map showing which pieces of code needed to be in memory at the same time and what their size requirements would be. Not surprisingly, the amount of memory available was far less than the amount needed. Had an evaluation of the limits been made at the beginning, the company would have realized that FORTRAN was going to be impractical for most of the programs in the system.

Documentation

The third step should have been to document the existing system. Using data flow diagrams, structure charts[8] or data structure charts, and Chapin charts,[9] the company could quickly have documented what had been done and could immediately have seen which parts of the system were usable and which needed to be thrown away.[10] This would also have fostered a team approach to maintenance and development rather than the severely individual, egocentric programming that occurred.

Determine staffing needs

The fourth step should have been to consider the expertise the company had and what they needed, as well as to determine how many people would be necessary for the project.[11] They tried to do this on a gut-feeling basis and started with five people, ending with 17. They did a fairly good job of matching expertise with needs when they added people, although they never had anyone who excelled in systems design, maintenance, or documentation. Since they had so many consultants, who were not going to be responsible for the maintenance function, there was no attempt at standardization, and no consideration was given to maintainability.

Set coding and design standards

The fifth step would be to set up coding and, as necessary, design conventions.[1,2]

Set up an implementation plan

The final step would be to set up an implementation plan with review procedures.[1] Had structured methodology tools been used, this step would be fairly obvious. A Gantt chart was finally developed in July, but until then no one knew what was expected. Since there were no regular status meetings and little communication, the chart was meaningless when it was put up, especially since there was no faith in the reasonableness of the estimates. The result was that several people sat around for many months not accomplishing much. One programmer sat from July to December before anyone wanted to see the results of his code. Predictably, when he ran his programs, it was discovered that they did not perform as required. This caused bad feelings and a significant loss of time.

The plan should have included an evaluation period for each section of software to determine the extent of modifications necessary.[1] By the end of that period a decision could be made about whether that section should be modified or rewritten. As it was, some people spent far too much time trying to work with existing software, whereas others looked at it for about 15 minutes before deciding (and always deciding) to throw it away.[10]

The project had too many strikes against it from the beginning to ever be really successful; but had it been handled correctly, it could have been termed a valiant and effective salvation effort.

SITUATION II—AN ACCEPTABLE SITUATION

Background Information

The second situation makes a far happier story. This one takes place at my current place of employment, Henry S. Miller Co., in Dallas, Texas. Henry S. Miller is a real estate and property management company. The company's data processing systems were developed on a leased IBM System/3 Model 10 during the period 1972 to 1976. In 1974 the process was contracted to a service bureau, which also assumed the hardware. In 1978 another company acquired the service bureau and assumed responsibility for providing processing and programming services to Henry S. Miller. The overall level of support and service recvied was not always satisfactory, so the company hired a local management consulting firm to review the support services. Over time the service requirements of users intensified to the point where an in-house computer was considered feasible. The consulting firm then helped Henry S. Miller write a request for proposal and select a hardware vendor.[12]

Despite the long association with IBM, a Data General computer was chosen. The choice was based on performance tests conducted, and it considered primarily current needs. The machine was delivered in late December 1981 and was operational in early January. (The staff, which consisted of a programmer/analyst and a nonprogramming data processing manager, had begun to use a computer at the Data General office starting in October.)

Because data processing was being done by the service bureau, no data processing staff was in place. A data processing liaison, hired in mid-1979, learned all about the systems Henry S. Miller was using. Shen then hired a programmer/analyst and became the data processing manager. The problem they faced was converting systems neither of them had written from IBM to Data General.

In December another programmer/analyst was hired, and a consultant who had Data General experience was engaged. The conversion schedule was tight. Everything was supposed to be converted by June 1982. Most of the systems were converted fairly close to their scheduled date, but there are still a few problems. The fact that they came close is amazing, considering that it has been said that "conversion estimates should not be attempted by persons who do not have access to a sizable data base of information and who do not do this on a regular basis."[13]

The biggest problem is documentation:[3] The systems are not well documented. There are a few system flow charts, and theoretically there are source listings for each program. Not only are the programs not printed one module per page; a program does not even necessarily start at the beginning of the page (see Appendix 1). Program names are not descriptive; therefore, which program goes with a given system has to be determined from the JCL, of which there are copious quantities (see Appendix 2). Unfortunately, this occurs in more cases than should be expected.[1, 15]

Deloitte, Haskins and Sells reviewed the accounting systems and recommended that they be completely rewritten. This firm felt that there was little point in trying to modify undocumented RPG programs. Because of time and staff limits, however, this was not possible.

I started with the company in May 1982 and immediately began lobbying for documentation to be done. The major drawback to this is user expectations.[1] Like many companies, we are operating in the crisis mode most of the time. Fast on the heels of the IBM-to-Data-General conversion has come a conversion of the manual and automated accounting systems to accrual from modified cash, which meant further changes to newly converted programs.

Understanding the Systems

To help myself understand the systems I was working on, I made a few data flow diagrams (DFDs). I found that in a relatively short period of time I became conversant with those systems. I have not made DFDs for most of the systems because I have not had time; consequently, their operation remains far fuzzier to me than the systems for which I have data flow diagram documentation.

Special Conversion Problems

A troublesome problem in the conversion was the conversion of files. IBM sign conventions for packed fields are different from Data General conventions,[14] and the records were not always created with a consistent sign character. The problem was not always immediately noticeable, and it caused problems that were hard to track. The conversion involved a painstaking process of looking at the data to see what was happening, and it consumed a lot of time.

Another problem was that the programs seemed to be written by adherents of the Magic School of Programming.[1, 13] That is, we were frequently heard to exclaim, "How could that ever have worked?" I'm sure if we spent long enough thinking about it, we could have figured it out; but frequently we just rewrote that piece. This problem seems to occur any time that someone else has written a program and has not followed strict coding standards.[13]

Scope of the Project

The primary directive, again, was to get the systems up and running as quickly as possible—in any way possible. The data processing manager gave the programmers a great deal of leeway in deciding whether to use an existing program; therefore, on most of the systems, a decision was made fairly quickly about whether to convert or rewrite a program. Thus, not a lot of time was spent trying to use unworkable programs, but some time was wasted in rewriting programs that probably could have been converted. This has not been a problem for us so far, but it needs to be watched in the future as rewrite decisions become more expensive.[10, 16]

Handling User Requests

One recurring problem has been users' wanting "just a little change—and how soon can we get it?"[3] This had come about because special user requests were taken into consideration as the systems were being converted. Users would come to each member of the data processing team with requests. As a result, some priority work was not getting done and user requests were being forgotten. The solution was to institute a special request form (shown in Appendix 3) and to channel requests through the data processing manager. This has been a tremendous help to us and is extremely important in dealing with inherited software, since what on the surface looks like a simple change could turn out to be extremely difficult because of the way the program is coded.[1] We are given the request and asked to estimate the amount of time it will take to make the change. On the basis of our estimates the requests can then be prioritized. As a result the users feel that they are getting better response to their requests.

Prospects for Design

Because the major effort has been the conversion of the systems, little design work has been done. Some of the systems run inefficiently, and others are organized in such a way that modifications are extremely difficult to make. There are also redundant programs and files; and because our storage space needs are very cyclic, we almost run out of space at some times and use only about half our disk space at others.

A thorough system design needs to be made. We need to get together with the users and find out everything that they would like to include in their systems.[10] The best way for us to do this is to make a set of data flow diagrams of the existing systems and then mini-models of the requested changes.[5] From this we can tell which changes can be made in the existing systems and which will mean total redesign and rewrite.

Comparison of the Two Situations

To compare the Henry S. Miller situation with that of the engineering company shows decidedly that Henry S. Miller was in a much better situation to begin with than was the engineering company. The expertise of the two managers was roughly equal, but at Henry S. Miller the manager is aware of her lack of programming knowledge and has found reliable people to depend on for advice. She has been open to suggestions about documentation and respects programmer decisions involving rewrites. This has added tremendously to department productivity.

In both cases a lack of documentation has hurt the implementation effort. In Henry S. Miller's case, however, at least the programs were demonstrably executing before the conversion effort began. In addition, at Henry S. Miller there is still the possibility of doing good documentation.

Both companies made use of consultants, and both to some extent gave up some control to the consultants. In both cases the consultants were conscientious; however, the Henry S. Miller consultant has made some effort to write programs that are readable and maintainable.

In general, Henry S. Miller started with a better situation, and through reasonable management it has kept the conversion and maintenance effort under control.

SUMMARY

To summarize, there are several considerations involved when a data processing department is dealing with inherited software. First, a relatively experienced staff is required.[2,3] The learning curve is high and fast, and the programmer doesn't have time to deal with language concepts.[16] Second, the staff needs to be virtually overwilling to do documentation, since most of the work involved in the conversion is documentation.[3, 16] Third, just as in new development, much front-end planning needs to be done, including implementation planning that specifies a time limit for evaluating existing software. This means that management has to accept the existing software as a sunk cost—if it works, great, but if not, rewrites are acceptable.[1]

ACKNOWLEDGMENTS

I would like to thank Patricia Cathey, my data processing manager, for supplying the background information and keeping me honest in my description of the Henry S. Miller situation.

I would like to thank Ned Chapin of InfoSci for his guidance and encouragement, without which this paper would never have been written.

I would like to thank LWFW, Inc., Group for the use of their word processor.

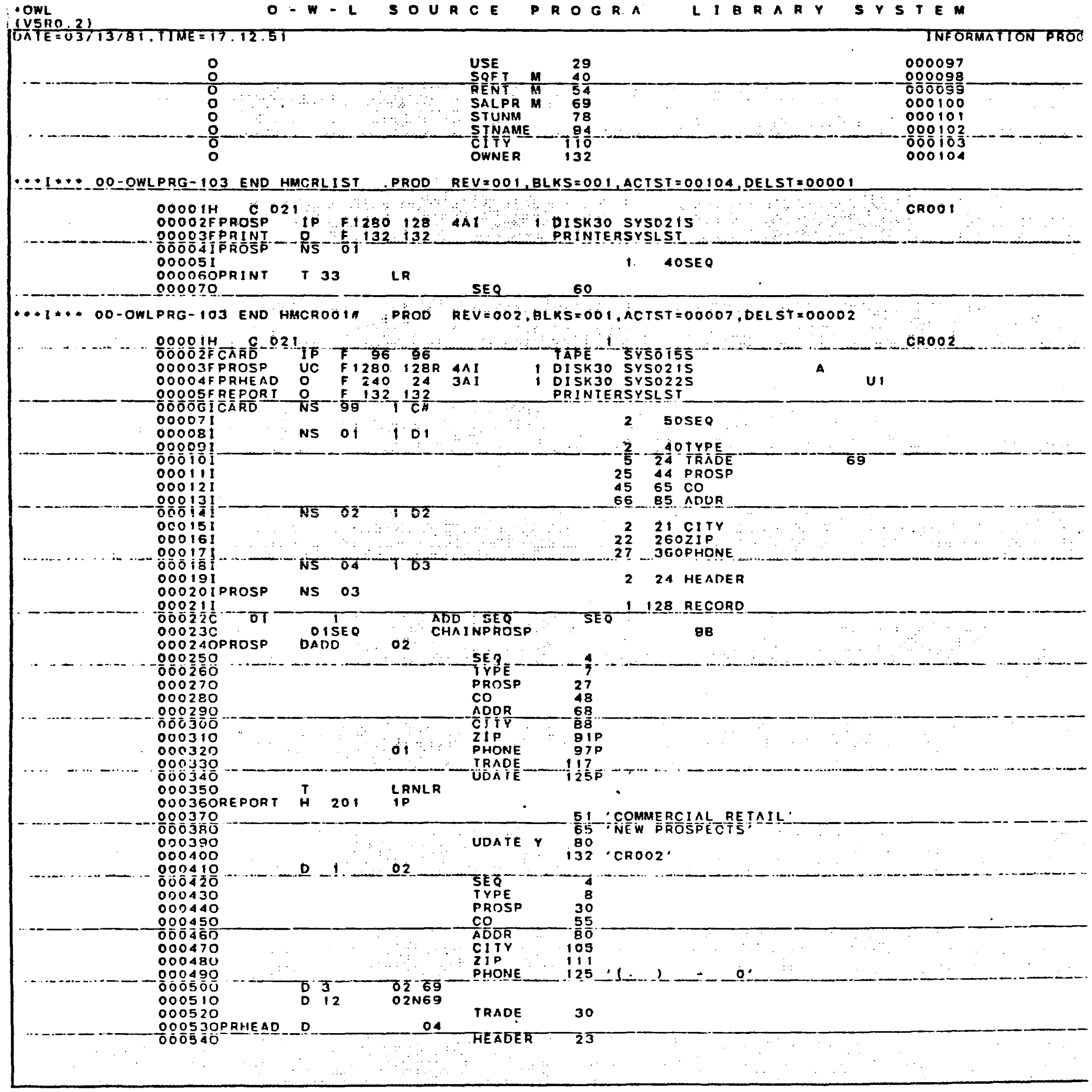
Finally, I would like to thank my husband Bruce for doing online literature searches on this topic and for his encouragement and suggestions.

REFERENCES

1. Liu, Chester C. "A Look at Software Maintenance." *Datamation,* 22 (1976), 51–55.
2. Parikh, Girish. "Three Is Key to Maintenance Productivity." *Computerworld,* 15 (1981), SR/40–SR/41.
3. Lientz, Bennet P., and E. Burton Swanson. *Software Maintenance Management.* Reading, Mass.: Addison-Wesley, 1980.
4. Parikh, Girish. "Cost-Effective Software Maintenance." *Interface: Data Processing Management,* 7 (1982), 37–39.
5. DeMarco, Tom. *Structured Analysis and Systems Specification.* Englewood Cliffs, N.J.: Prentice-Hall, 1979.
6. Gane, Chris, and Trish Sarson. *Structured Systems Analysis: Tools and Techniques.* Englewood Cliffs, N.J.: Prentice-Hall, 1979.
7. Orr, Ken. *Structured Requirements Definition.* Topeka, Kans.: Ken Orr and Associates, Inc., 1981.
8. Page-Jones, Meilir. *The Practical Guide to Structured Systems Design.* New York: Yourdon, 1980.
9. Chapin, Ned. "Semi-Code in Design and Maintenance." *Computers and People,* 27 (1978), 17–27.
10. Lyons, Michael J. "Salvaging Your Software Asset (Tools Based Maintenance)." *AFIPS, Proceedings of the National Computer Conference* (Vol. 50), 1981, pp. 337–341.
11. Clark, David M. "Maintenance Programming." *Computerworld,* 14 (1980), 27–30.
12. LWFW, Inc., Group. *Henry S. Miller Companies, Data Processing Request for Proposal.* Private company document, May 1981.

13. Parikh, Garish. *How to Measure Programmer Productivity.* Chicago: Shetal Enterprises, 1981.
14. Hyman, Dale. Private communication, August 1982.
15. Chapin, Ned. "Productivity in Software Maintenance." *AFIPS, Proceedings of the National Computer Conference* (Vol. 50), 1981, pp. 349–352.
16. Glass, Robert L. *Software Maintenance Guidebook.* Englewood Cliffs, N.J.: Prentice-Hall, 1981.

APPENDIX 1

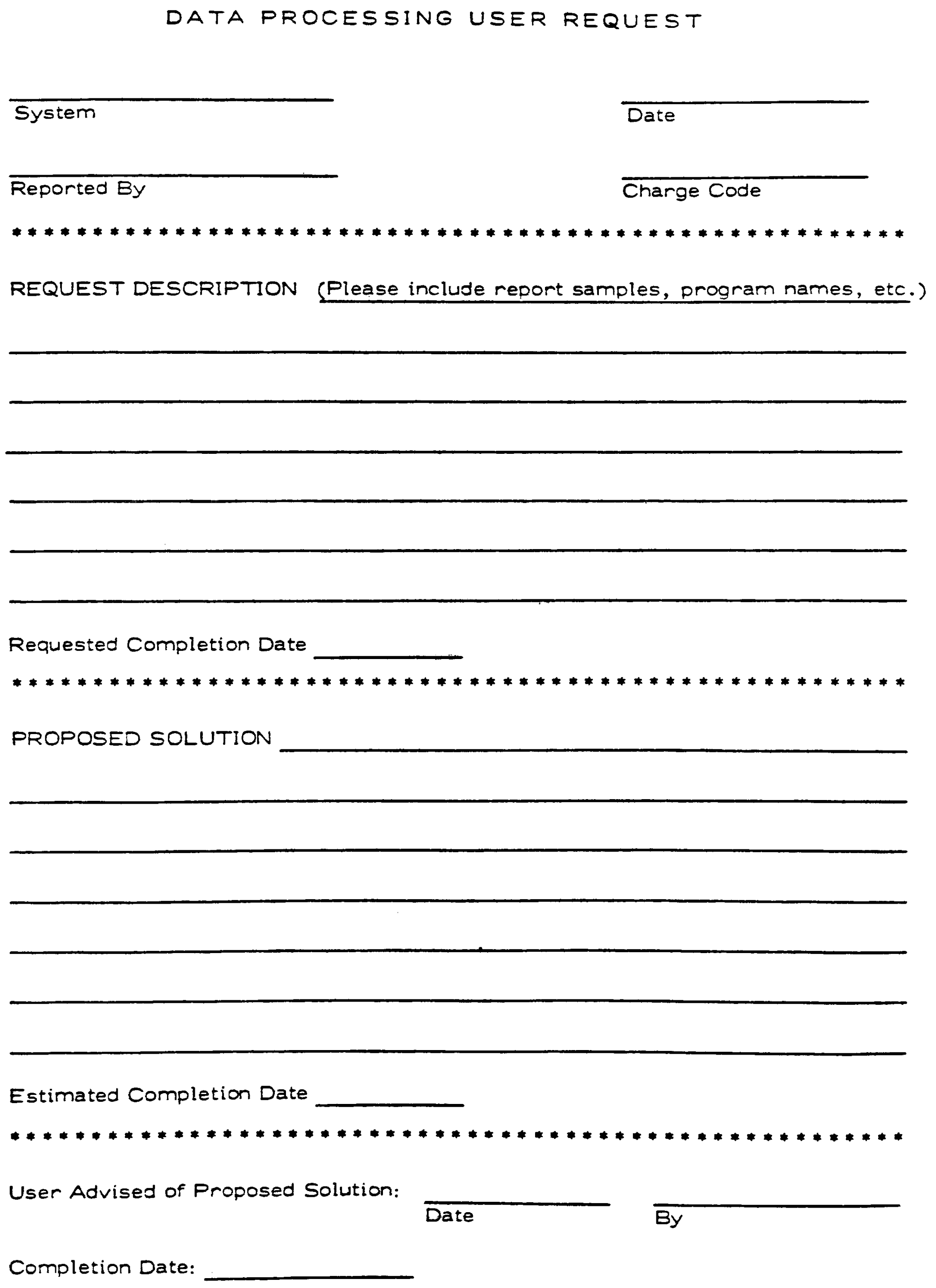


```
*OWL                  O - W - L   S O U R C E   P R O G R A   L I B R A R Y   S Y S T E M
(V5R0.2)
DATE=03/13/81,TIME=17.12.51                                                    INFORMATION PROC

            O                          USE      29                             000097
            O                          SQFT   M 40                             000098
            O                          RENT   M 54                             000099
            O                          SALPR  M 69                             000100
            O                          STUNM    78                             000101
            O                          STNAME   94                             000102
            O                          CITY    110                             000103
            O                          OWNER   132                             000104

***I*** 00-OWLPRG-103 END HMCRLIST  .PROD  REV=001,BLKS=001,ACTST=00104,DELST=00001

          00001H    C 021                                                      CR001
          00002FPROSP   IP  F1280 128  4AI      1 DISK30 SYS021S
          00003FPRINT   O   F 132 132             PRINTERSYSLST
          00004IPROSP   NS  01
          000051                                       1   40SEQ
          00006OPRINT   T 33      LR
          000070                       SEQ      60

***I*** 00-OWLPRG-103 END HMCR001#  .PROD  REV=002,BLKS=001,ACTST=00007,DELST=00002

          00001H    C 021                          1                           CR002
          00002FCARD    IP  F  96  96             TAPE   SYS015S
          00003FPROSP   UC  F1280 128R 4AI      1 DISK30 SYS021S          A
          00004FPRHEAD  O   F 240  24  3AI      1 DISK30 SYS022S              U1
          00005FREPORT  O   F 132 132             PRINTERSYSLST
          00006ICARD    NS  99   1 C#
          00007I                                       2   50SEQ
          00008I        NS  01   1 D1
          00009I                                       2   40TYPE
          00010I                                       5  24 TRADE        69
          00011I                                      25  44 PROSP
          00012I                                      45  65 CO
          00013I                                      66  85 ADDR
          00014I        NS  02   1 D2
          00015I                                       2  21 CITY
          00016I                                      22  260ZIP
          00017I                                      27  360PHONE
          00018I        NS  04   1 D3
          00019I                                       2  24 HEADER
          00020IPROSP   NS  03
          00021I                                       1 128 RECORD
          00022C    01      1        ADD  SEQ       SEQ
          00023C         01SEQ      CHAINPROSP                98
          00024OPROSP   DADD     02
          000250                       SEQ       4
          000260                       TYPE      7
          000270                       PROSP    27
          000280                       CO       48
          000290                       ADDR     68
          000300                       CITY     88
          000310                       ZIP      91P
          000320                 01    PHONE    97P
          000330                       TRADE   117
          000340                       UDATE   125P
          000350        T         LRNLR
          00036OREPORT  H  201    1P
          000370                                51 'COMMERCIAL RETAIL'
          000380                                65 'NEW PROSPECTS'
          000390                       UDATE Y  80
          000400                               132 'CR002'
          000410        D  1     02
          000420                       SEQ       4
          000430                       TYPE      8
          000440                       PROSP    30
          000450                       CO       55
          000460                       ADDR     80
          000470                       CITY    105
          000480                       ZIP     111
          000490                       PHONE   125 '(   )   -    0'
          000500        D 3      02 69
          000510        D 12     02N69
          000520                       TRADE    30
          00053OPRHEAD  D           04
          000540                       HEADER   23
```

APPENDIX 2

```
P.HMGLADDS     V.M 0.0         17 BLOCKS         PRIVATE SOURCE-STATEMENT LIBRARY

                    BKEND   P.HMGLADDS

          * $$/* CYCLE HMGLADDS   ADDS GENERAL LEDGER TRANSACTIONS        08-28-81
          // PAUSE    MOUNT  **  H S M M S T  **  ON ANY MOD 1
             ASSGN SYS006,DISK,VOL=HSMMST,SHR
          // EXEC IPRESTRT
          * $$ DATA RESTART
          * $$/*
          // RESET ALL
          *  JOB HMGLKEY1      TRANSACTION KEYTAPE                  RESTART=YES
          * $XLST CLASS=U,DISP=D,COPY=01,CMPACT=IP00,FCB=$$BFCB                A
          // PAUSE  MOUNT KEYTAPE  'HMKTGLO2'  ON X'287'
          // ASSGN SYSLST,IGN
          // ASSGN SYS007,X'287'               INPUT HMKTGLO2
          // TLBL EPKEYTP,'&01WORK1,W'         OUTPUT
          // EXEC LSKEYTP
          * $$ DATA HMKTGLO2
          * $$/*
          // RESET ALL
          *  JOB HMGLO25*        ADD TRANS TO WORK FILES             RESTART=YES
          * $XLST CLASS=U,DISP=D,COPY=01,CMPACT=IP00,FCB=$$BFCB                A
          // TLBL CARD96,'&01WORK1,R'          INPUT
          // DLBL GLWRK1T,'==CPU--TEMP.C1'     HMGLWRK1-OUTPUT
          // DLBL GAPDETT,'==CPU--TEMP.C2'     HMGAPDET-OUTPUT
  1°      // DLBL GAPDETI,'HMGLGAPD',0,SD      INPUT
          // EXTENT SYS006,,1,0,2432,0209
          // DLBL GLWRK1I,'HMGLWRK1',0,SD      INPUT
          // EXTENT SYS006,,1,0,2033,0399
          // EXEC HMGLO25*
          * $$/*
          // RESET ALL
          *  JOB HMGLSRT1     RECREATE HMGLWRK1 MASTER               RESTART=YES
          * $XLST CLASS=U,DISP=D,COPY=01,CMPACT=IP00,FCB=$$BFCB                A
          // ASSGN SYSLST,IGN
          // DLBL SORTIN1,'==CPU--TEMP.C1(D)' HMGLWRK1           SRT 1
  2°      // EXEC SORT
             OPTION PRINT=CRITICAL,FILNM=(GLWRK1),SORTOUT=6
             SORT FIELDS=(1,13,BI,A,20,8,BI,A),WORK=1
             RECORD TYPE=F,LENGTH=80
             INPFIL BLKSIZE=800
             OUTFIL BLKSIZE=800
             END
          * $$/*
          // RESET ALL
          *  JOB HMGLSRT2     RECREATE HMGAPDET MASTER               RESTART=YES
          * $XLST CLASS=U,DISP=D,COPY=01,CMPACT=IP00,FCB=$$BFCB                A
          // ASSGN SYSLST,IGN
          // DLBL SORTIN1,'==CPU--TEMP.C2(D)' HMGAPDET
  3°      // EXEC SORT
             OPTION PRINT=CRITICAL,FILNM=(GAPDET),SORTOUT=6      SRT 2
             SORT FIELDS=(1,14,BI,A,20,8,BI,A),WORK=1
             RECORD TYPE=F,LENGTH=80
             INPFIL BLKSIZE=800
             OUTFIL BLKSIZE=800
             END
          * $$/*
          // RESET ALL
          *  JOB HMGLBKUP      WESTINGHOUSE BACKUP                  RESTART=YES
          * $XLST CLASS=U,DISP=D,COPY=01,CMPACT=IP00,FCB=$$BFCB                A
          // EXEC DSCASSGN
          // ASSGN SYS000,DUMMY
          * $$/*
          // TLBL SYS000,'HMGLBKUP,U'
          // ASSGN SYS003,SYS006
          // UPSI X11
          // EXEC COPYDT
             GLBAL.IR
             GAPBAL.IR
             GLMST.IR
             GLBUD.IR
             GLCOMPN.IR
             DEPARTM.IR
             GLWRK1.SD
             GAPDET.SD                                                              75
          * $$/*
          * $$/*      END OF JOB
```

APPENDIX 3



DATA PROCESSING USER REQUEST

System ____________________ Date ____________________

Reported By ____________________ Charge Code ____________________

****

REQUEST DESCRIPTION (Please include report samples, program names, etc.)

Requested Completion Date ____________

****

PROPOSED SOLUTION ____________________

Estimated Completion Date ____________

****

User Advised of Proposed Solution: ____________ Date ____________ By

Completion Date: ____________

Maintaining user participation throughout the systems development cycle

by RANDY J. RAYNOR
and
LINDA D. SPECKMANN
Texas Instruments
Dallas, Texas

ABSTRACT

Effective user participation is well known to be an important aspect of good system development methodology. Specific tools and techniques for managing user participation in all phases of the systems development life cycle are illustrated by a large business system development project at Texas Instruments. Emphasis is placed on maintaining a constant level of communication between user and developer as the system design evolves.

INTRODUCTION

During the past few years there has been an increased awareness of the importance of user involvement in every phase of systems development.[1] The approaches to achieving effective involvement vary from new tools usable by the end user to full-time assignment of a user to assist in the development process. Most of these techniques have some drawbacks because the right kind of tools and personnel are not available.

Many of the tools require either extensive training time or expensive application software. The user assigned to a full-time systems job is too often one who is not vital to the business operations. This user may not be the best source of information for the system developers and, because of detachment from the project, has no direct authority to implement the completed system in the project.

This paper draws from recent experiences at Texas Instruments with the design of a large product configuration management system to illustrate the selection of a requirements definition methodology easily understandable by the user community, as well as techniques for continued user participation throughout detail design, construction, and testing. A brief overview of the initial phases of the systems development life cycle is presented, and additional information is cited from a prior paper.[2] The focus of this paper is on the later phases of the life cycle. The tools and techniques outlined require little training and no application software, allowing easy implementation in other environments.

Case Study Background

The need for a new configuration management system was widely recognized at Texas Instruments, especially for the highly complex radar and guidance products developed for the Department of Defense. The development was chartered as a part of the Engineering Information System—an implementation of IBM's Administrative Engineering Information Management System (AEIMS) concept.[3]

At Texas Instruments, development of new products is organized through projects operating like small companies, with a staff for engineering, drafting, and configuration management as well as other design support functional areas. There was no companywide configuration management organization or focal point through which to coordinate development of a functional requirements specification. Rather, there were a large number of independent configuration managers with an equally large number of approaches to product configuration management. Although some small systems did support configuration management functions, they were not widely used.

The configuration management system was required to control and report product baseline evolution from concept formulation through production sustaining, including product drawing and specification development, change approval and tracking, data management, product as-built verification, and logistics support. Thus, this system would affect not just those independent configuration managers, but also every other discipline involved in product development and manufacturing.

THE SPECIFICATION APPROACH

Historically, systems developers have often considered user involvement a necessary evil. With the configuration management system it is assumed that the user is the subject matter expert, the user's involvement and support in every phase of system development is required, the user's time is a valuable commodity, and without the user's commitment the system will probably fail. This assumption places a heavy responsibility on the team of users (which must effectively represent all users), and it explains the importance of techniques for efficient use of team resources.

During the initial analysis of the configuration management system and team identification, it became evident that the players had widely differing opinions on what configuration management was and how it should be accomplished. This problem prompted an approach to system specification different from that typically followed.

System specification was broken into three distinct phases: concept definition, functional requirements, and detail design. The concept definition specification identified at a high level how configuration management should be accomplished. The functional requirements specification identified in more detail what functional steps and data were required to achieve the basic operations of the concept definition. Neither of these two specifications made reference to the computer systems solution. The detail design specification then identified which functions should be computerized and gave the exact definition of the system design from the user's perspective.

Selection of a Specification Methodology

The selection of a system specification tool was based on the following requirements. It had to

1. Be easy for users to learn, easy to use, and easy to change.
2. Serve as an effective communication tool between systems analysts and users.
3. Support the top-down decomposition of concept/function/detail design specifications.

DeMarco's data flow diagrams[4] were selected. They had only a few simple syntax rules, which could be learned in an hour. The 8½ × 11-inch size of each diagram was also convenient for team review because an overhead projector could be used.

Figure 1 represents the use of data flow diagrams to support the decomposition from the concept specification to the design specification. The concept specification defined the top three to five levels of the data flow hierarchy. The functional specification added an additional one to three levels to the data flow hierarchy where required, until specific user tasks were identified. Then a functional task description was developed for each. The design specification mapped the functional tasks to system modules and defined the design of these modules.

CONCEPT DEVELOPMENT AND REVIEW

The concept definition document defined the fundamentals of configuration management and the necessary procedures to accomplish them effectively. The following topics were covered:

1. Concept Definition Overview—The configuration management development history and the purpose, scope, and introduction of the concept definition
2. Introduction to Configuration Management Concepts and Definitions—A definition of the major configuration management functions
3. Present Operations—A discussion of systems and procedures in each area
4. Data Flow Diagrams—The diagrams depicting the product life cycle
5. Benefits—A discussion of expected benefits from the implementation of a system
6. Appendix—A list of key users, discussion of existing systems, and a keyword glossary

Twenty members of the user community were selected by a company vice-president to participate in the review of the concept definition. The team was composed of representatives from 10 functional areas, with emphasis on configuration management, quality assurance, logistics, engineering, and manufacturing.

To use the large team in the best way, the concept definition data flow diagrams were divided for review so that each set had a primary effect on one functional group. Subteams of about six members were formed, each led by an expert in the functional subject matter.

The product of the review was a 73-page document that described the overall approach to the way configuration management should be performed throughout the life cycle of a TI product. More important, however, the result of the review was a good working relationship between developer and user, based on mutual respect, and an appreciation by users and management for the effort that would be required over the next several years.

FUNCTIONAL REQUIREMENTS DEVELOPMENT AND REVIEW

The objective of the functional requirements document was to record data flow and process requirements down to the task level. The specification addressed the following topics:

1. Functional Requirements Specification Overview—The objective and scope of the functional requirements document
2. Configuration Management Overview—A brief discussion of the fundamentals as defined in the concept definition
3. Configuration Management Subsystem Descriptions—An overview, set of data flow diagrams, and function descriptions for eight major areas
4. Configuration Management System Development

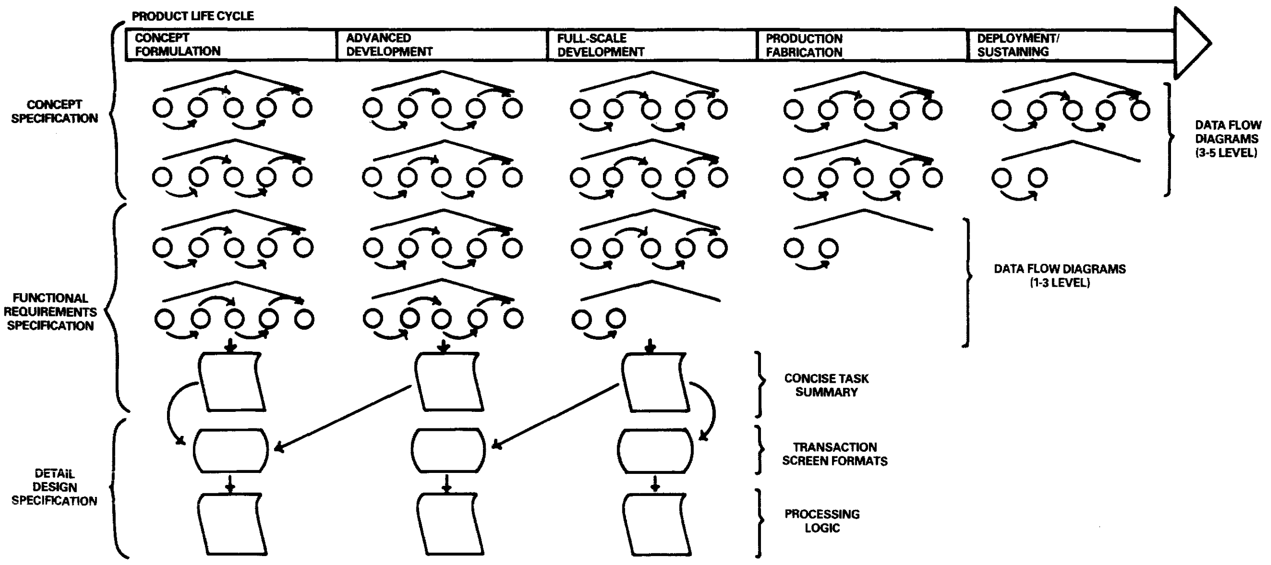


Figure 1—Specification approach

Methodology—An explanation of the development life cycle
5. Appendix
   a. Data flow diagrams and descriptions
   b. Survey trip reports
   c. Logical database views
   d. Data element glossary
   e. Keyword glossary
   f. Configuration management forms
   g. Review team minutes

The draft functional requirements specification defined additional levels of data flow diagrams below those of the concept definition, emphasizing areas for potential system application. Task descriptions were then developed to define further many of the processes in the lower levels of the data flow diagram. The task descriptions defined an operation in terms of the input and output data and a set of processing steps.

The functional specification review team was partially selected from the membership of the concept review team. Those retained were experts in their fields and had schedules that permitted participation. Other new members were added to strengthen representation from functional areas that were greatly affected by configuration management tasks. This review demanded a more detailed understanding of data associated with daily tasks and therefore required representation from the users with a working knowledge of the function.

The result of this review was an 1100-page document that defined the complete set of tasks required to accomplish configuration management. From this list, tasks could be selected for detail design.

Since the scope defined by the functional requirements specification would require 3 to 4 years to develop, an initial phase was defined for detail design and construction. The data flow diagrams developed in the functional requirements specification played an important role in the first phase of detail design as well as subsequent phases. They provided the complete set of requirements used to prioritize later system releases.

The relatively small scope of the initial phase not only provided near-term system startup, but also improved user involvement and motivation by providing a goal requiring project level planning for implementation. The phased schedule is illustrated in Figure 2.

DETAIL DESIGN SPECIFICATION

The detail design specification defined the user interface to the application system, procedures for use of the system, overall software architecture, and database design. The contents of the detail design specification were as follows:

1. Introduction—The purpose and scope of the detail design specification document
2. System Description—An overview of the entire system's functional design
3. Subsystem Description—Data flow diagrams of all system functions, procedural flows and narratives, and online transaction descriptions
4. Changes to Existing Procedures—A step-by-step procedure describing manual and system operations
5. Test Plan—Test objectives and methodology
6. Training Plan
7. Appendix—Logical database design, manual forms, review team minutes, and glossary

Because of the detail required in the design specification, data flow diagrams were insufficient to represent the required information. The strength of the data flow diagram is its hierarchical illustration of a major problem. When a detail design is being made, procedural flows, control points, feedback loops, and organizational responsibilities become important. Therefore, procedural flows were chosen to show how the system would be used in day-to-day operations. However, as

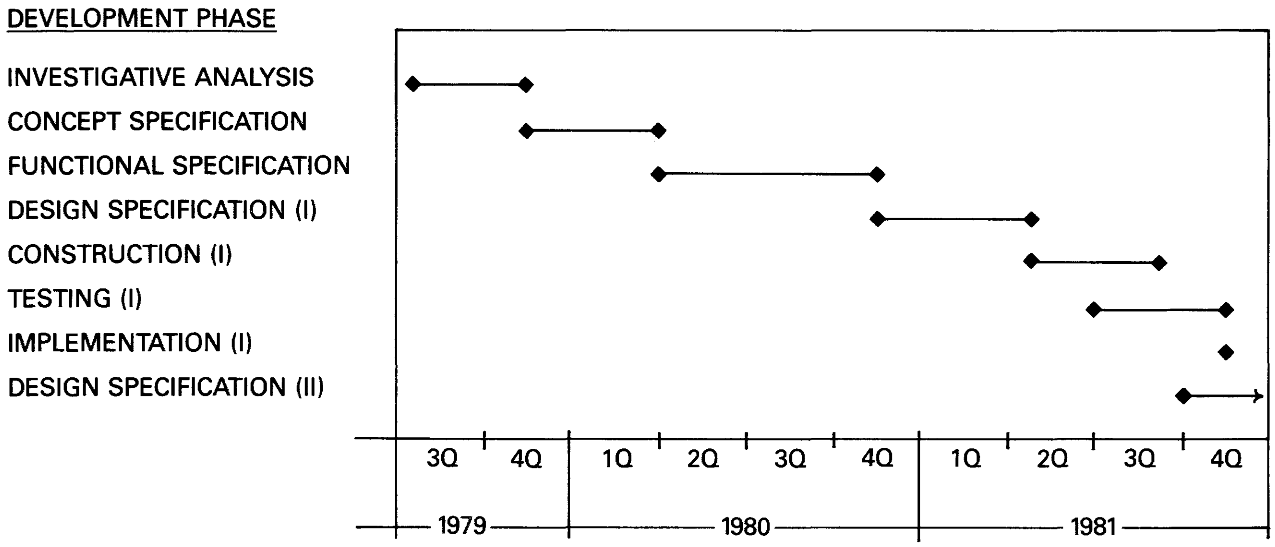


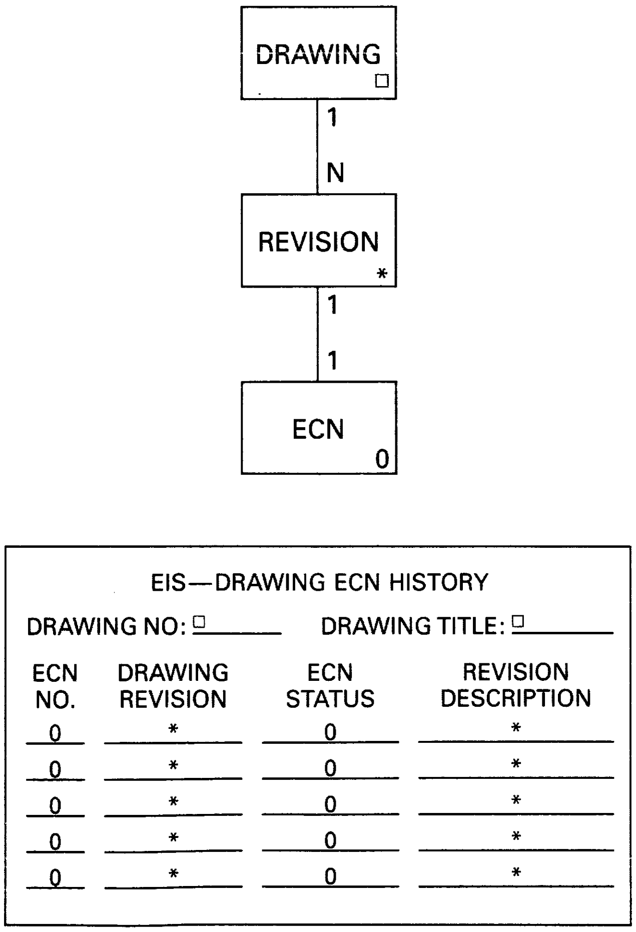
Figure 2—System life cycle

each phase of detail design was initiated, the data flow diagrams were consulted for information on interfaces required within the configuration management system as well as for other systems.

User Review

The review team consisted of 12 members of the user community representing five disciplines. Some of these members had been on the functional requirements team, but new team members were needed to challenge the work of the earlier team.

As is typical in system design reviews, the format and usage of each online transaction was described and discussed in detail. User procedures, including manual and system operations, were also presented during the eight-session review. Since the application was to be developed using a database management system, the database design was reviewed with users. Figure 3 illustrates how users were involved in such a system-oriented topic as database design. The database design was presented by using colored dots relating each data item on every screen to its location in the database. The users were then asked to evaluate the database design to insure that



EIS—DRAWING ECN HISTORY

DRAWING NO: □ DRAWING TITLE: □

ECN NO.	DRAWING REVISION	ECN STATUS	REVISION DESCRIPTION
0	*	0	*
0	*	0	*
0	*	0	*
0	*	0	*
0	*	0	*

Figure 3—Database review

1. All items of importance (entities) were identified.
2. All attributes of each entity were identified.
3. All relationships (one-to-one, one-to-many, many-to-many) between entities were properly identified.
4. Potential future requirements would fit within the overall structure.

During the database review several attributes were found to be associated with the wrong entities, but the overall structure remained unchanged. The users did, however, express comfort at having a better understanding of this hidden aspect of system design.

Design Verification

After the design review there was doubt that the design had been reviewed thoroughly enough to be certain it would function in an operational environment. The concern was due to the significant changes required in manual procedures, coupled with the users' lack of experience with the use of computer systems. An additional test was added to the design process to validate the approved design further.

The design verification test manually simulated the system operation. The test goal was to determine whether all necessary data had been captured and the proposed manual procedures would work. Six areas were chosen to participate; some representatives were not from the detail design team. By using paper versions of the proposed online transactions, users were asked to operate the system in parallel with their current procedures. Although execution of this test was initially sporadic, the systems analyst was able to provide sufficient motivation to obtain a complete set of test results. The results are shown in Table I.

Development Plan

Before the system was constructed, the complete schedule was developed for the construction and user test phases of system development. The format of the schedule was such that the users would understand not only what was going on within the development organization, but also their responsibilities during the development cycle. At this time specific users were identified to assist in the construction and test

TABLE I—Design verification test results

	NUMBER IDENTIFIED	PER CENT OF TOTAL	NUMBER IMPLEMENTED	PER CENT OF TOTAL
DESIGN PROBLEM	17	17%	16	19%
DESIGN IMPROVEMENT	32	33%	26	31%
PROCEDURAL PROBLEM	20	20%	17	21%
PROCEDURAL IMPROVEMENT	9	9%	6	7%
TRAINING PROBLEM	19	19%	16	19%
TRAINING IMPROVEMENT	2	2%	2	3%
TOTAL	99		83	

phases, and the first group was identified for initial production startup of the system. Joint commitments were made by developer and user—the developer to meet the specification and the schedule and the user to implement the system on a project. User responsibility for the system had shifted from the user community at large to the specific users' own operating organization.

SYSTEM DEVELOPMENT

The construction phase consisted of six major milestones for each transaction being developed. First, a preliminary design review was conducted between the analyst and the assigned programmer to insure that details of the design specification and the overall systems architecture were understood. The next step was the development of the user documentation for the transaction by the programmer.

A review of user documentation was then conducted between the developer and user identified in the development plan. This was a key review to insure that the programmer understood the design specification as well as any implications not detailed in the design specification. It also gave the user a final chance to make modifications to the design prior to coding, and it established a working rapport between the programmer and the user. (Though user documentation is seldom written before coding, the method is widely recognized as a good one. In addition to the benefits mentioned above, it maintains user involvement at a point in the development cycle that is typically void of any involvement. In this case the maintenance of involvement was extremely beneficial in preventing the common attitude among users that after the specification is complete, user participation stops.)

The approved draft of the user documentation was sent to the technical writing staff for editing while the programmer continued with the construction phase. The next step was program design and pseudocoding, which concluded with a program design review. After source coding was completed, a code walkthrough was conducted with other members of the programming staff. Upon approval by the lead programmer the assigned programmer completed testing of the program and preparation of the test package for the user unit test. This test package consisted of the completed user documentation and transaction test instructions.

User Test

The test package was used by the user identified in the development plan to help in understanding the transaction and the purpose of the unit test. This test averaged a cycle time of less than 2 weeks for each transaction.

Once all programs had passed the unit test, the users conducted a system integration test to insure proper communication of data among all the transactions within the system. This test required about 2 weeks to complete.

The final test was the parallel production test, which, unlike prior tests, used real data in the operational environment. Before beginning the parallel production test all users participated in a training class, which allowed a prototype test of the training material. The parallel production test lasted for about 6 weeks.

During the entire testing cycle, weekly meetings were held in the user's area to review test results. All comments were documented for future disposition. User reaction to the testing was favorable. Users felt they could affect the design of the production system and therefore were motivated to complete the testing. The success of these efforts is presented in Table II, which displays the program changes identified during design as compared to those identified during construction and test. The results show that the majority of changes were identified during design, indicating a successful communication of the design between the developers and the users.

Implementation and Fanout

The development plan was executed on schedule, and the system was put into production with no major problems. After a month of use on the first project area, the system was installed in a second area. Several minor problems arose during the first month of use on the second project. However, users from the first project volunteered to meet with users from the second project to assist them in several procedural issues. This cross-communication between users was helpful in achieving user acceptance of the new system and procedures.

Several months into system operation a major procedural issue was identified by one of the functional user groups indirectly affected by the system. The issue should have been resolved during the review of the detail design specification. It was discovered that the user who represented the functional area affected had attended less than half of the design specification review meetings. Although this issue was resolved without any major impact, the occurrence did point out that particular attention must be paid to insuring that the users accept the *responsibility* that goes along with the *authority* they have as team members representing their respective user communities.

RESULTS

During the concept and functional requirements specification activities, data flow diagrams were a key in providing a struc-

TABLE II—Design changes by module

MODULE	LINES OF CODE	CHANGES DURING DETAIL SPECIFICATION USER REVIEW	CHANGES DURING CONSTRUCTION AND TEST
1	2638	25%	none
2	1855	5%	5%
3	1646	100%	0%
4	1987	60%	10%
5	1251	15%	none
6	1117	100%	1%
7	1074	not in spec	100%
8	2637	30%	1%
9	2173	100%	none
10	4336	80%	10%
11	3467	100%	5%

ture to maximize effective use of the users' time. Beginning with the detail design specification, use of procedural flow and control flow diagrams replaced the data flow diagrams. Following the user review of the design specification, several ad hoc techniques were used to maintain a constant level of user involvement and interest during the construction and test cycle. These techniques included design verification test, development planning, user documentation walkthrough, and unit and parallel production test.

The techniques communicated to the users progressively more detailed perspectives of the system design, allowing them to visualize and critique the system. The frequency of these milestones permitted the users to monitor the progress of the development activity. As the design evolved, the users saw the incorporation of their ideas into the system design and felt more sure that the system would succeed in solving the problems in their environment.

The 50 to 60 users involved in the various development activities had a major stake in the successful implementation of the system. As shown in Figure 4, the user contribution was 30% of the total effort and was spread across all phases of development. Since their involvement was voluntary (and in addition to their regular job), the amount of their contributions illustrates their acceptance of responsibility for their role in the development activity. As the system design evolved, this responsibility grew—to such a degree that the users felt that the system was theirs by the time of implementation.

CONCLUSION

Tools and techniques for systems development should be selected to support frequent communication between the developers and the users. It is not the particular tools that are important, but the selection of a set of tools that improve the users' understanding of the design as it evolves.

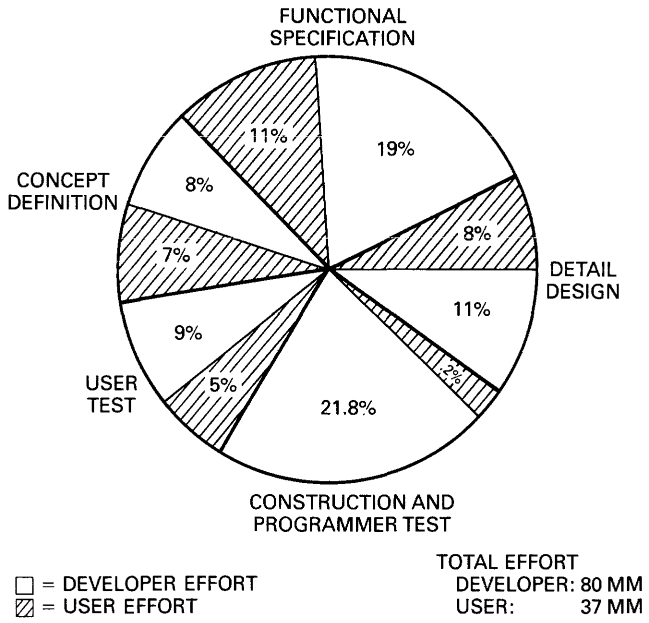


Figure 4—System development effort by phase

REFERENCES

1. Jackson, James E. "The Role of the User at Standard Oil Company (Indiana) In The Development of Large-Scale Business Systems." *AFIPS, Proceedings of the National Computer Conference* (Vol. 51), 1982, pp. 549–553.
2. Raynor, R. J., and L. D. Speckmann. "Structured User Participation in Systems Requirements Specification." In *Proceedings of the 26th Annual Conference of the Society for General Systems Research on Systems Methodology*. Louisville, Ky.: The Society for General Systems Research, 1982, pp. 305–310.
3. IBM. *Administrative Engineering Information Management System: System Definition*. New York: IBM Corporation, 1975.
4. DeMarco, T. *Structured Analysis and System Specification*. New York: Yourdon, 1979.

Data processing project management: A practical approach for publishing a Project Expectations Document

by LOIS ZELLS
Yourdon, Inc.
Scottsdale, Arizona

ABSTRACT

With the mounting demand for proficient personnel and the parallel increase in salaries, management is seeking ways to improve productivity in order to realize a higher return on their investment dollars. Knowing what to do, when to do it, and how to do it prevents costly retries. Given any kind of a project and 2 to *N* participants, there will be 2 to *N* views of the project. Furthermore, there are always dozens of subtle nuances floating like little puffs of smoke over every enterprise, and they are often not in agreement. It is necessary to crystallize the assumptions that each participant "understands" to be the accepted expectations, resolve the conflicts, and disseminate this information to the community. This paper reflects a practical method for transforming facts and conflicts into an approved development approach and publishing the results in what will be called a Project Expectations Document.

INTRODUCTION

Never underestimate the importance of managing organizational expectations! In any data processing project, effective presentation of the many agreements and decisions is a must!

Made during informal interviews and conversations on the phone, in the hall, and by the elevators, as well as in formal meetings, these agreements and decisions need to be documented. This record starts as soon as the project is initiated and is continuously revised throughout the development cycle.

A project, any project, is composed of four basic stages:

1. Definition of the problem
2. Design of the solution
3. Development
4. Implementation

It is during the definition stage, before the "real" job of analysis, that developers have produced what has traditionally been called the statement of work, feasibility study, or survey.

In its simplest form, a feasibility study may be an economic evaluation such as a cost/benefit analysis based on some high-level assumptions. A more complete definition recognizes that analysis of the existing system deficiencies and new system objectives is required as input to the survey. The likely output, the statement of work, may include the following:

1. A cost/benefit analysis
2. A narrative of the project and its deliverables
3. A high-level project plan
4. Preliminary solution alternatives
5. The recommended approach

Generally accepted essentials of the survey, the listed components meet the goals of a feasibility study. However, there are also many *undefined* and *assumed* attitudes and expectations infused into every development effort. *N* number of people will have *N* number of perspectives pertaining to the project, and they are usually not all in agreement. *It is necessary to crystallize these views, resolve the conflicts, and disseminate this information to the community.*

This paper proposes to restate the obvious, shift activities, integrate new ideas, and repackage the product. A practical approach will be provided for transforming the old, incomplete feasibility study into a comprehensive preanalysis phase that illuminates the variations of expectations and concludes with the publication of a Project Expectations Document.

A PRACTICAL APPROACH FOR PUBLISHING A PROJECT EXPECTATIONS DOCUMENT

What is so special about a Project Expectations Document (PED)? Why do we even want to publish a PED? The participants, who range from executive management to the hands-on operators of the system, need to have a clear understanding of the project and its deliverables. This clarification must occur as soon as the project is initiated and must be maintained throughout the project development cycle.

The PED will record all the philosophies, assumptions, dependencies, requirements, and constraints. Organizational expectations will be realistic, approved, documented, and disseminated. Furthermore, the red-flag issues will be brought into the open and dealt with immediately. Failures occur when project participants know that risks are being increased while visibility is being suppressed because management cannot (or will not) acknowledge the problems. However, even the most taciturn managers cannot refute facts that are clearly stated.

To Begin

The foundation for building a PED is the interview process. Information is gathered informally and formally. Informal interviews occur during phone conversations, at the water cooler, during lunch, and so forth. A notebook carried at all times to record the essentials of every conversation helps to identify individual expectations and conflicts. Thus, assumptions, dependencies, constraints, and philosophies are quickly accumulated and documented.

Procedurally, the *formal* fact-gathering process is much easier. At least three to four organizational levels should be interviewed, using a predeveloped questionnaire and allowing time to record the entire meeting after each interview. The interviewee should be allowed to verify and/or correct the interview report. After all the interviews have been recorded, reviewed, and revised, the results should be analyzed and summarized according to areas of agreement and conflict. Disagreements can be resolved in committee or by a selected group or individual. The original areas of agreement, the conflicts, and the resolutions constitute the final document.

Publication of the PED Requires Time and Effort

During this preanalysis Project Expectations Phase, the fact-gathering process just described is employed. A consolidated document is the deliverable, and the activities leading up to its completion are as follows:

1. Publish a draft PED.
2. Select the reviewers of the draft PED.
3. Distribute the draft PED.
4. Schedule the review of the draft PED.
5. Review the draft PED.
6. Revise the draft PED.
7. Publish the PED.

Some of these jobs must be done sequentially; others can be done in parallel. Partitioning publication into smaller tasks and their dependencies allows better control and management of the entire process.

A PED Addresses Many Issues

It is impossible to develop a generic solution to all projects in all environments. Therefore, the following list with the subsequent discussion is offered as a menu of suggested areas of interest:

1. Executive Overview
2. Scope of the project
3. Systems overview
4. Analysis of alternatives
5. Project management philosophies
6. Constraints, assumptions, and dependencies
7. Costs
8. Completion criteria
9. Success criteria
10. Acceptance testing criteria
11. Project plans
12. Management reports
13. Management summaries

Executive Overview

Senior managers are too busy to read a lengthy document. They want the information in an easy-to-absorb offering. Although it is the last job completed in the phase, the one-page Executive Overview is the first item in the document and describes the project history, the existing system efficiencies and deficiencies, the new system goals, the costs of the new system, a cost/benefit overview, and the estimated completion date.

Also included in the Executive Overview is a brief description assessing the effect on existing operations of doing/not doing the project. Impact assessment will be discussed in detail in the following section.

Scope of the Project

In this section of the PED the boundaries of the new system are set. The user organization is established, identifying the affected departments and describing their major areas and components. We decide what is to be built, what needs to be done, and how long it should take. It is also necessary to specify what the system will not do and to determine existing manual and automated systems.

Impact assessment

With the introduction of the impact assessment, a helpful dimension is added to the Scope of the Project section. Often, as project managers, we assume that our project is pure goodness, something akin to motherhood and apple pie. How can users fail to love and embrace it? The project may be completed on time and within budget, yet the users are unhappy. Why? Can the project then be considered a success?

If there is any chance of resistance, the situation should be assessed early in the project. Perhaps the condition can be overcome (publicity, user involvement, and so forth); but if the project will have a truly negative impact on the organization, let us know about it now. If we cannot correct the problems, do we want to spend $*N* on a white elephant?

The Impact Assessment answers such questions as these:

1. How do the users feel about the new system?
2. Where are the resistance pockets?
3. Will operations change?
4. Will departmental boundaries change?
5. Will jobs be changed? Added? Eliminated?
6. How will users react to new technology?
7. What is the priority of the project in relationship to existing business operations?
8. What is the effect of the project on existing business operations?

Evaluating these issues early in the project enables informed and effective decision making.

Systems Overview

Although the project expectations phase is a preanalysis phase, it is nevertheless necessary to initiate some high-level analysis at this stage. The Systems Overview explains the current system's deficiencies, but in-depth study is postponed until the analysis phase. The key features of the new system are identified, new functions are described, and development costs are estimated in order to justify, prioritize, and recommend those that may be included, delayed, or excluded. The major inputs and outputs to the system are identified, and system interfaces are declared. Performance requirements may also be included, along with any pertinent comments on existing hardware or software.

Analysis of Alternatives

The project team is responsible for providing management with several solutions or proposals for the new system. Immediately following analysis there should be a special phase for in-depth evaluations of several implementation alternatives. However, some foundation work may begin during the project expectations phase. For each alternative, summarize the key features, major functions, assumptions, advantages/disadvantages, opportunities/risks, and estimated costs. Recommendations are also appropriate.

Project Management Philosophies

The desire to manage projects effectively and be perceived as exceptional contributors stimulates our interest in many issues, not the least of which is how much control and responsibility is to be assumed and by whom. Relying on the psychological adage that involvement implies commitment, we propose that the selection and approval of the project management philosophies be a participative activity for the following items:

1. Development life cycle,
2. Planning philosophy,
3. Problem management plan,
4. Formal plan for review,
5. Change management plan,
6. Approval cycle plan,
7. Organization plan, and
8. Status reporting plan.

The tasks described in the earlier sections, *To Begin* and *Publication of the PED Requires Time and Effort,* should be applied to producing a document for each item in this list.

Development life cycle

Traditional project development usually follows some set of steps from start to completion. The steps may be a series of ad hoc responses as needs are recognized, or the project may be completed using an accepted methodology. Furthermore, a given project may use only a limited number of steps chosen from the methodology. More enlightened data processing environments may have a selection of several kinds of methodologies, which enables the customization of a project development life cycle. It is unusual to solicit management and user involvement in the choice of the development approach. Nevertheless, that is precisely what we are advocating.

Planning philosophy

We must help the organization to understand that project planning is an iterative process. It is impossible to present a comprehensive and detailed schedule for implementation on the first day of a project. Furthermore, it is unlikely that a project plan that is precise can be completed before design is finished. Consequently, as we migrate through the development life cycle, our knowledge base of the project becomes broader and we are able to refine the plan continually.

We will also advise the organization to employ the participative approach (those who are closest to the work will plan and estimate); and we will tell the organization the method of partitioning the work and the way in which we calculate dates.

Problem management plan

No project can go from start to completion without its share of problems. Contentions build gradually and often perniciously. Usually it is not until a situation has become serious that its resolution becomes an issue. We recognize that problems are unavoidable and will institute a problem management plan.

Formal reviews

The procedures and participants for walkthroughs and formal reviews are identified.

Change management plan

Change is inevitable. Any time we attempt to freeze a specification, we are deceiving ourselves. We are only restricting the system's view of the real world. Admitting that we cannot control change, we will manage change by implementing easy-to-follow procedures.

Approval cycle

The final document from each phase must be approved, the project plans must be approved, the analysis specification must be approved, the design must be approved, and so on. It is advisable to itemize all the activities in the project that need ratification. A responsibility matrix is completed; it designates accountability and provides visibility to important activities.

Organization plan

The organization plan describes the partitioning of the project into specialty groups, the group charters, the assignment of participants to groups, and the group members' percentage of participation.

Status reporting plan

If a project control and accounting system is already installed in the environment, it is important to indicate which reports will be used, who will be on the distribution lists, and how often the reports will be produced.

If the organization has no reporting system, an evaluation of alternatives (which should include manual reporting, purchased packages, and internally developed software) should be presented, along with a recommended plan of action.

Constraints, Assumptions, and Dependencies

It is important to specify the subtle nuances that are usually floating like little puffs of smoke over every project. There are dozens of little assumptions that everyone "understands" are the accepted expectations of the project. Rarely, however, are these subtleties addressed until they become issues:

1. User participation is clarified.

2. The four classic tradeoffs when a project is late are delineated.
3. Computer requirements such as equipment availability and turnaround time are specified.
4. Responsibility issues such as levels of control, skill levels, and conflict resolution are stated.
5. Technical decisions pertaining to software packages, design tools, and development tools are made.
6. Management overhead is detailed.
7. Administrative support is defined.

Costs

The determination of costs is needed for informed decision making, evaluation against benefits, budgeting, and establishing the yield on the investment.

Costs to date and projected costs to completion will be reported as each phase is completed so management can make informed go/no-go decisions to proceed.

Development costs may be weighed against the benefits of anticipated operating savings, high returns, improved service, or tax savings.

The payback period for return on investment, net present value, and internal rate of return may be used to compare the yield on the investment against other potential ventures.

In our continued effort toward managing organizational expectations, it is important to stress that estimates at this stage may be misjudged by as much as 70% to 200%.

Completion Criteria, Success Criteria, Acceptance Testing Criteria

It is important to stipulate, early in the project, the conditions for determining that the project is finished, that the project is successful, and that the project is acceptable.

Agreeing on a terminating landmark prevents project completion from floating toward infinity. But how will we know when the project is finished? Will the conclusion of an activity such as acceptance testing or the shakedown period indicate completion? Will the system be considered delivered after it has run error-free for some prescribed period of time? The criteria for terminating the project must be specified now.

Success criteria may be described from varying perspectives. Management, users, operations, and data processing each have their respective standards for success:

1. Is the project on time and within budget?
2. Is the system user-friendly?
3. Does the system operate efficiently?
4. Is the system easy to maintain?
5. Is the system an asset to the organization?

These and similar questions will be addressed in identifying the requirements for success.

Acceptance testing is the act of simulating a live environment with conditions that will exist after Day 1 of implementation. It is the final process that satisfies the users that the system is operational.

Although the development of the actual test cases for acceptance is done in the testing group after the analysis phase, it is important to give the process of acceptance testing visibility early in the project. The level of user participation and a demonstration plan will be defined at this time.

Project Plans

At the conclusion of the project expectation phase, the project may be continued or canceled. If the decision is made to carry on, the next phase will be the analysis phase. During the analysis phase a parallel group of activities for planning project completion will also be occurring. Restated, the next events that occur will be analysis *and* planning.

The PED will contain three sets of plans:

1. A detailed plan for analysis
2. A detailed plan for planning
3. A high-level plan for project completion after analysis

Management Reports

Major milestones, major responsiblities, and preliminary resource requirements are identified and high-level network diagrams such as PERT/ CPM or Gantt charts are drawn.

Management Summaries

Accumulating historical information on project development serves as a foundation for repeating successes and avoiding failures. Problems and their resolutions, possible pitfalls, and successful approaches are described in summaries of the current project expectations phase and the project history to date, and a narrative of the planning effort is written.

IN CLOSING

To document organizational expectations takes time and it takes people! Managers who resist dedicating time and resources to this project early are deceived into believing the effort will not be expended later in reacting to undefined expectations. A construction project would not be launched without defining the method of building as well as what was to be built. If it were, there would be misunderstandings and the necessity for demolition and reconstruction—time-consuming and costly—or living with the error. The same analogy can be brought to project planning.

In attempting to introduce change in your organization, study your subject, know your facts, and provide supporting documentation. Cite problems in your environment and benefits to be derived from the new approach. Develop an action plan and provide recommendations. Choose a pilot project and establish a time and dollar range. Plan and schedule an orientation meeting and

GOOD LUCK!

ACKNOWLEDGMENTS

This paper has been written on airplanes, in hotel rooms, and other places where it is virtually impossible to carry reference material. Therefore, aside from using the outline from one chapter in the Workshop for Project Planning and Control that I developed for Yourdon, Inc., this paper is pure "stream of consciousness."

Nevertheless, I would like to acknowledge that I could never be in this place without the help of all my professional colleagues, who forced me to crystallize my thoughts.

That I bear the scars of managing many projects without managing expectations legitimized my underlying belief, "There has to be a better way." I am grateful to the organizations that allowed me to test my hypotheses.

I am also grateful to the literature (see the following list) that supports my belief and to the company that allows me to proselytize.

BIBLIOGRAPHY

1. Yourdon, E. *Managing the Structured Techniques.* New York: Yourdon Press, 1979.
2. Yourdon, E. *Managing the System Life Cycle.* New York: Yourdon Press. 1982.
3. Metzger, P. W. *Managing a Programming Project.* Englewood Cliffs, N.J.: Prentice-Hall, 1973.
4. Burrill, C., and L. Ellsworth. *Modern Project Management.* New Jersey: Burrill-Ellsworth Associates, Inc., 1982.
5. Thomsett, R. *People and Project Management.* New York: Yourdon Press, 1980.
6. Peters, L. *Software Design: Methods & Techniques.* New York: Yourdon Press, 1981.
7. Brooks, Jr., F. P. *Mythical Man Month.* Reading, Mass.: Addison-Wesley, 1972.
8. Myers, G. J. *The Art of Software Testing.* New York: John Wiley & Sons, 1979.
9. Dickenson, B. *Developing Structured Systems.* New York: Yourdon Press, 1981.
10. Page-Jones, M. *The Practical Guide to Structured Systems Design.* New York: Yourdon Press, 1980.
11. Wiest, J. D., and F. K. Levy. *A Management Guide to PERT/CPM.* Englewood Cliffs, N.J.: Prentice-Hall, 1977.

DATABASE/DISTRIBUTED SYSTEMS

Donald R. Hyde
IBM
San Jose, California

As we move into the mid-80s we see major advances in database technology. The sessions in this track will touch on the subjects that seem to offer the greatest potential in the future:

A close look will be taken at what database management system options exist for the microcomputer user through the distribution of databases as parts of a large central data processing operation.

New database technologies such as the much-awaited relational database systems will be explored, along with the often-discussed database machine for unloading the central processing complex.

In our session on enterprise analysis, we will also take a look at the process one goes through in deciding how to describe business requirements in terms translatable into database systems.

A distributed database design for a communications network control system

by S. C. LO, S. L. KOTA, and M. H. ARONSON
Ford Aerospace & Communications Corporation
Palo Alto, California

ABSTRACT

A three-level distributed database concept is proposed to meet the high performance requirements of an integrated communications network control system. The three levels are source, user, and control. By using fully replicated subject databases in the source level, one regional operations center can serve as a backup for any other center. The applications database in each user level is a subset of the subject databases residing in the source level. Hence, the user level is localized and can respond to near real-time network control requirements. The control level coordinates updates to the distributed database copies. Interactions among the three levels are discussed and four operational stages are considered: initialization, continuous update, global synchronization, and data recovery. A broader usage of the data dictionary is proposed. Simplification of the replicated database design in terms of isolation of global and local data and some implementation considerations are provided.

INTRODUCTION

With the growing demand for telecommunications services, network control of the communications resources has become vitally important. Network control consists of network management (planning and analysis), resource control (asset allocation and configuration), traffic control (congestion control and routing), and technical control (performance and status monitoring).

A comprehensive distributed database management system for the communications network control system is essential if the system is to react successfully to contingency situations. This applies to commercial communications networks, and even more so to military networks. A communications network control system can be organized hierarchically by geographic regions (see Figure 1). A system operations center collects data for history and trend analysis, and also performs network control for communications links that cross one or more regional boundaries. The system operations center is connected to each regional operations center. Data regarding network configuration, status, and traffic flow are reported up to regional centers from the operating units; decisions, actions, and controls are dispersed down for execution. The operating units are the control positions located at the transmission media facilities (e.g., satellite communications terminals, voice switch facilities, etc.). The regional operations centers must also be able to provide database information to all other regional centers. Whether this is accomplished by some form of ring or star architecture depends on requirements for reliability, the number of regional centers, their location, and the associated recurring communications costs. In this paper, full connectivity between all control nodes is assumed.

Certain conditions exist in a network control system (particularly one used in a military environment) that place special requirements on the distributed database. The architecture is classically hierarchical. The regional operations center controls all operating-unit resources within its geographic area. Status flows up to the regional center and control flows down to the operating units providing regional autonomy. If the regional center becomes isolated from the rest of the control system, local operation within the region will continue. If the regional operations center experiences a prolonged outage (severe equipment failure, power failure, flood, etc.), another center must take over control of the operating units in the area. The operating units will communicate with the backup regional center via dial-telephone connections. Further, any regional center can back up any other center (not just nearest neighbors) to maximize survivability of the control capability. Backup operation must occur concurrently with control processing for the normal geographic region assigned to that center. This implies that a regional operations center normally uses only a small portion of the system database (that associated with its assigned region); however, under contingency conditions, that portion of the database normally used by another regional center is also accessed. To minimize software development costs and to minimize database configuration management problems, the entire system database is available at the system operations center and at each regional operations center. Only parts of this database are used by a given center for real-time operations. However, data pertinent to operation of any other center is available should backup operation be necessary. As described later in this paper, the real-time localized database is separated from the global database to avoid deadlock conditions during updates. The local database is implemented by creating region-peculiar views of the database components pertinent to a given center.

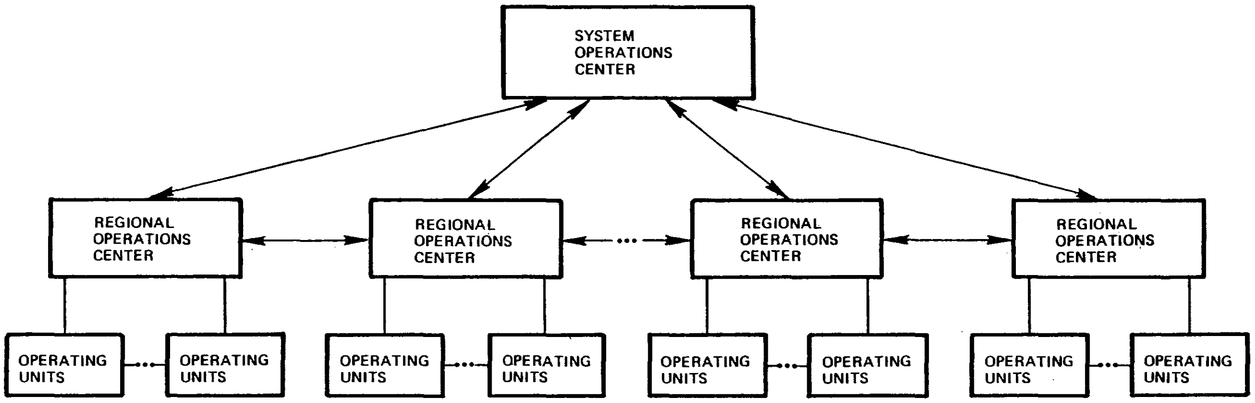


Figure 1—Hierarchical network control system

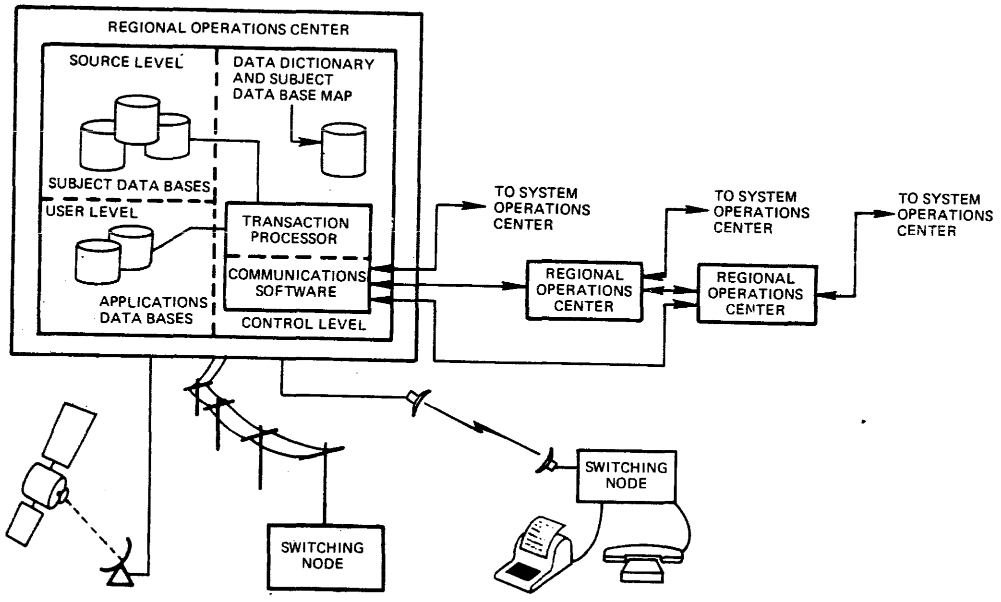

REGIONAL OPERATIONS CENTER
SOURCE LEVEL
DATA DICTIONARY AND SUBJECT DATA BASE MAP
SUBJECT DATA BASES
USER LEVEL
TRANSACTION PROCESSOR
COMMUNICATIONS SOFTWARE
APPLICATIONS DATA BASES
CONTROL LEVEL
TO SYSTEM OPERATIONS CENTER
REGIONAL OPERATIONS CENTER
TO SYSTEM OPERATIONS CENTER
REGIONAL OPERATIONS CENTER
TO SYSTEM OPERATIONS CENTER
SWITCHING NODE
SWITCHING NODE

Figure 2—An integrated communications system

To be optimally responsive to the traffic and configuration requirements placed on the communications system at any given time, it is necessary to allow all of the regional control facilities to access current information on the communications assets served by that center. It has been reported[1,2] that by using replicated databases (i.e., all the control facilities have the same copy of the database), the near real-time functions such as traffic monitoring and control can be performed satisfactorily. Sanli et al. discusses control of one type of communications asset, satellite communications systems.[1] The distributed traffic database proposed for the Japanese Telecommunications Network supports only one network control function, namely, traffic control.[2] This paper addresses control of a multimedia communications system with a wide range of functional capability.

Owing to the mixed-media transmission systems involved in an integrated communications system (e.g., terrestrial telephone, line-of-sight microwave, troposcatter, and satellite communications), the network control system described here has to perform several complex control functions. These include resource control, traffic control, and technical control for data and voice transmission using different types of media. Because of the complexity of an integrated network control system, the traffic between the control facilities becomes much heavier than the systems referenced in the literature. Mohan provides an excellent outline of the research done so far on distributed database management.[3] The issues that are associated with distributed processing[4] and, in particular, those that deal specifically with replicated databases[5,6,7] have put constraints on the network control database design.

In brief, the constraints are

1. Currency of the data. If the database is concurrently modified by more than one control facility, there is a danger that the database will be left in an ambiguous state. For the network control application, this situation becomes more serious because (a) Another control computer may need current information to back up a failed regional control center, and (b) The communications and protocol delays may severely impede the performance of the near real-time functions.
2. Data recovery. Data may be lost because of the failure of data storage devices (e.g., a disk crash), loss of data communications, or incomplete database update. For the network control application, an additional constraint is put on data recovery because new updates may be flowing in while recovery is taking place. A different mechanism, other than using the approach of completing one data transaction after another,[4] must be implemented to retain the consistency state of the global information.

Advantages of using duplicate copies of the same database at all regional control facilities are

1. Each regional control facility can serve as the backup for any other failed center.
2. Up-to-date data are readily available to permit restoration of operation at a control center after failure.
3. Near real-time functions have better performance because all necessary data are locally available.
4. Maintenance of the database becomes simpler owing to identical data definitions and schema throughout the system.

This paper proposes a multilevel database model that consists of the source, user, and control levels applicable to a realistic network control problem. Mohan also discusses a

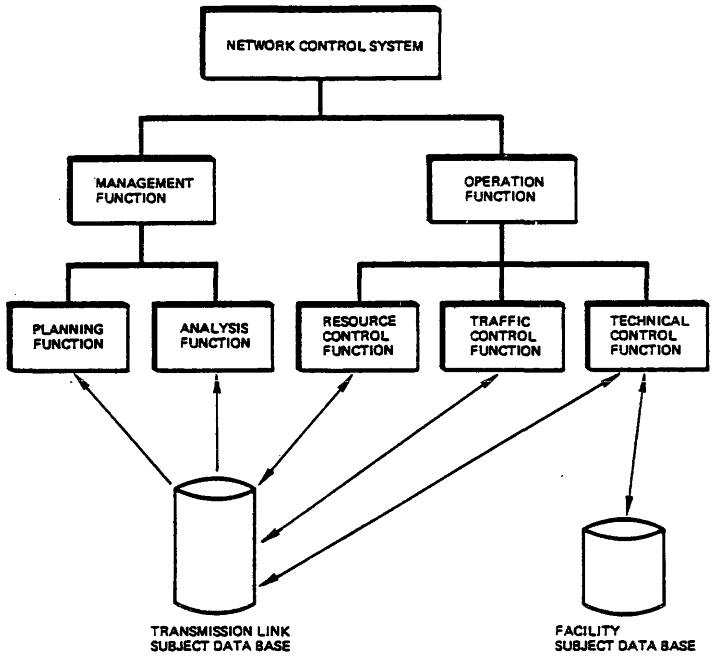


Figure 3—Subject databases for interaction with network control elements

multilevel database design along with a critical evaluation of the topic.[8] The source level contains the global database replicated at each center. The user level contains the real-time database (region peculiar) that is a subset of the source-level global database. The control level permits updates received in the user database to be propagated upward to the source level and to the control level at other centers in case they ever need to perform backup control. This architecture attempts to maximize the advantages just discussed, so that better system control performance can be achieved, and it reduces the impact of the constraints imposed on the distributed database.

In the next section a distributed database design concept is discussed in more detail. Besides being used as a documentation tool, the data dictionary becomes an important element of the control level to maintain global synchronization of the distributed database. The data flow between the three levels of the database at local and remote regional control centers is discussed. The subsequent section provides implementation considerations.

DISTRIBUTED DATABASE DESIGN CONCEPT

Communications Network Model

Figure 2 illustrates a control network for which a distributed database design concept is proposed. As shown in Figure 2, the digital transmission media may include telephone lines, line-of-sight microwave, tropospheric scatter, and satellite lines. The network control system consists of several fully-connected regional operations centers. The regional operations centers are functionally equivalent, maintaining common capabilities and databases. They differ in that they control assets in a specific geographic area and may vary in required control functions. One given regional operations center may have control over terrestrial and line-of-sight communications links while another may have additional control over a satellite communications system. All centers are capable of controlling all types of media; only the media in the center's geographic area are actively controlled.

Typical functions of a network control system may be divided into two categories: network management and network operation (see Figure 3). Planning and analysis are the major functions in network management. Through planning and analysis, the control system is capable of deciding where, when, and how to optimally use existing communications assets and how to integrate (hypothetically or actually) new facilities. Near real-time network operations include technical, traffic, and resource control. The technical control function includes testing, coordinating, and monitoring of communications assets to assure quality and continuity of communications. Given a transmission link that is identified by its origin, termination, and type of equipment, this function has the capabilities to

1. Perform loop tests between the originator and the terminator
2. Obtain status of the communications equipment (e.g., multiplex equipment)
3. Switch in and out, either manually or remotely, faulty or degraded equipment

In short, the technical control function monitors and maintains the health of the communications network.

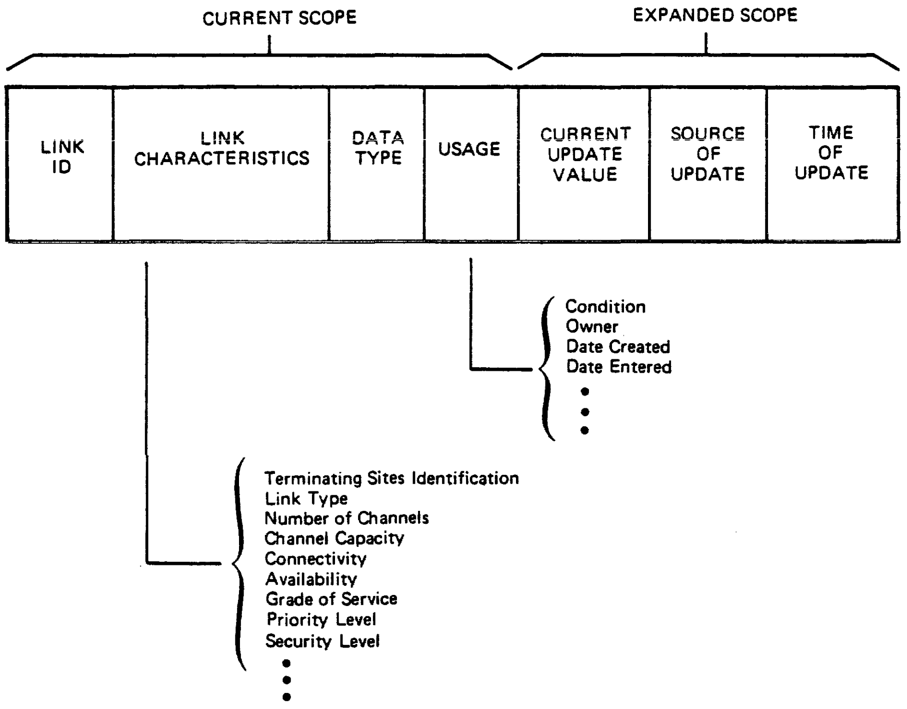


Figure 4—An example of the transmission link subject database data dictionary

Traffic control is a response (usually in the order of seconds) to traffic congestion on specific communications links and media. Traffic paths are dynamically rerouted based on routing algorithms.[9] Resource control is traffic control in a broader sense. The analysis and planning functions analyze and define a long-term traffic pattern. This configuration is passed on to the resource control function for implementation.

Assume that the transmission system has five voice-grade links, each of which is a standard frequency division multiplex (FDM) modulation group, that is the group is formed by mixing each of 12 voice channels with a particular carrier frequency associated with the channel. If one of the voice links (12 voice channels) is fully occupied, the traffic control function may distribute the incoming telephone calls to the other four trunk lines. If a new voice link is added to the transmission system, it becomes the responsibility of the resource control function to reconfigure the traffic pattern.

To support these network control functions, schema of the database may logically be organized into entities such as transmission link, facility status, switch connectivity, and alternate routing.

Figure 3 also shows two of the many subschemas that are used by different applications (functions). A subschema is a tailored view of a specific database schema and is considered to be a subject database. Therefore, a subject database is an entity that supports application processing functions such as planning and analysis. The transmission-link and facility databases (two of many) are included in the diagram to illustrate how subject databases interact with individual functions. An example of data entries for the transmission-link subject database data dictionary is shown in Figure 4. Because of the nature of regional control, data values for the entries are unique for every regional control center. Terminating-site identifiers in Figure 4, for example, are the names of switching nodes and are uniquely located at a certain geographical area inside a given control region and, by definition, are different from region to region.

Database Configuration

The database supporting the system operations center and the regional operations centers will have three levels (Figure 2): the source, user, and control levels.

The source level consists of a complete set of subject (global) databases. Data in this level are fully duplicated and synchronized among all regional operations centers to permit backup control operations.

The user level consists of a subset of the subject databases that is retrieved from the source level and becomes the applications (real-time) databases used by the various control processing functions for a specific regional operations center. Assume that one regional operations center does not perform the technical control function. Therefore, when this particular regional operations center is brought up for service, the applications databases at that center will not include the facility database for that region (which is associated with the technical control function). In other words, the user level is localized to support the activities of a specific center.

Note that if a specific subject database is required at an operations center to perform a network control function, the

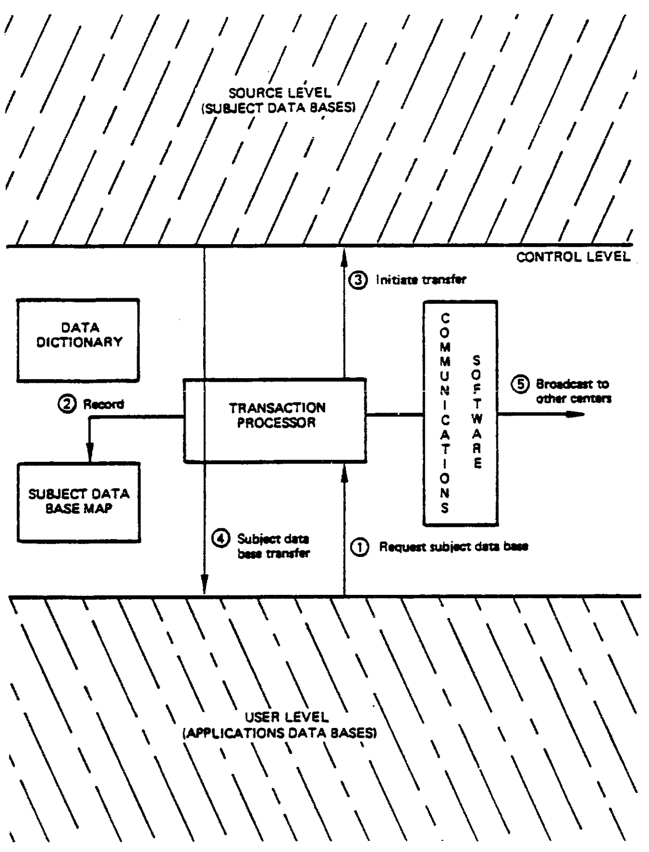


Figure 5—Database initialization

control level creates a view constraint on the user level so that only that data in the applications database that is pertinent to the region is accessible (for read and write operations). Real-time changes in the applications database at one regional center do not require updates to the applications databases at other regional centers, since the other centers normally ignore (have no view of) those sections of the applications databases. The subject databases, however, are updated at all centers as described later in this paper (to permit backup operation). This permits standard relational database management systems to be used in the source and user levels, simplifies the creation of the applications database copies, and restricts the authority for changing a specific data value to only one operations center (avoids deadlock situations).

The control level consists of four components: the transaction processor, data dictionary, subject database map, and communications software. The transaction processor coordinates the data flow among the components of the control level and between the user and the source levels. Traditionally, a data dictionary has been used as a documentation mechanism.[10] Generally, it lists all data items that are used, their definitions, how and where they are used, and who is responsible for them. However, the data dictionary may be expanded to include what data are being updated, as well as when and from where legitimate data updates may come. Figure 4 provides an example of such an expanded data dictionary. The data dictionary becomes the bookkeeping tool for global data updates. Although the data dictionary is capable of recording where data reside geographically, the subject database map is more effective to record this type of information. When a regional operations center is requested to back up another center, it is much faster to search through the subject database map to get the appropriate subject databases pertinent to the expanded area rather than to use the data dictionary. The communications software handles the actual data communications among the regional operations centers and the system operations center.

Data Flow

Data flow among the three levels of the database and among the operations centers can be illustrated in terms of four operating states: initialization, continuous update, global synchronization, and data recovery.

Initialization

The flow of events that occurs when a specific regional operation center is first brought up is illustrated in Figure 5. The user level requests, via the transaction processor, that the source level provide a number of subject databases. This information is recorded on the subject database map and the change in the map is passed on to the other regional operations centers. Meanwhile, the source level initiates transfer of the subject databases to the user level.

Continuous update

When the regional operations centers are in operation, the applications databases are constantly updated to reflect the current situations at the various operating units within the region. Therefore, there are two streams of data coming into the control level as shown in Figure 6. From the local user level, data are directly fed to the transaction processor. Data that are being updated at other centers (which affects the subject databases at all centers but not the applications databases) are received via the communications software. Standard data communication protocols, for example the International Standards Organization (ISO) model, may be used for this purpose.

Protocols such as ADCCP, HDLC, SDLC, and so on, may be used at the link level to ensure that updates are correctly received by each control center. Broadcast from the regional center that incurred the applications database change is used so that the same information can be disseminated to all centers. Before the data are recorded in the data dictionary, the data elements are checked against the subject database map for validity. This step seems to be redundant; however, this procedure ensures that data are coming from a legitimate source and have not been distorted because of transmission errors or because of any other source of errors in a secured database environment. At this point permanent changes to the global (subject) databases have not yet been made.

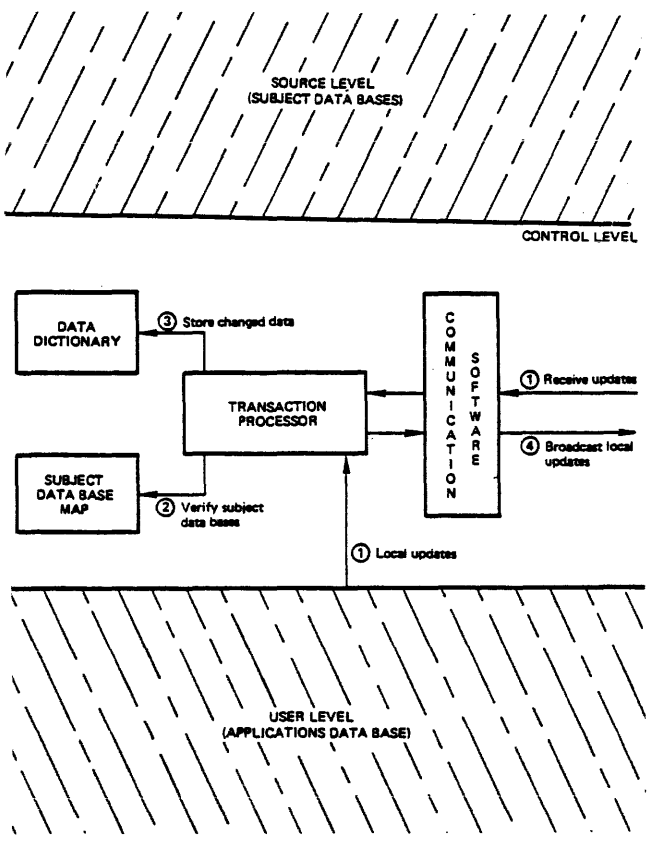


Figure 6—Continuous global and local data update

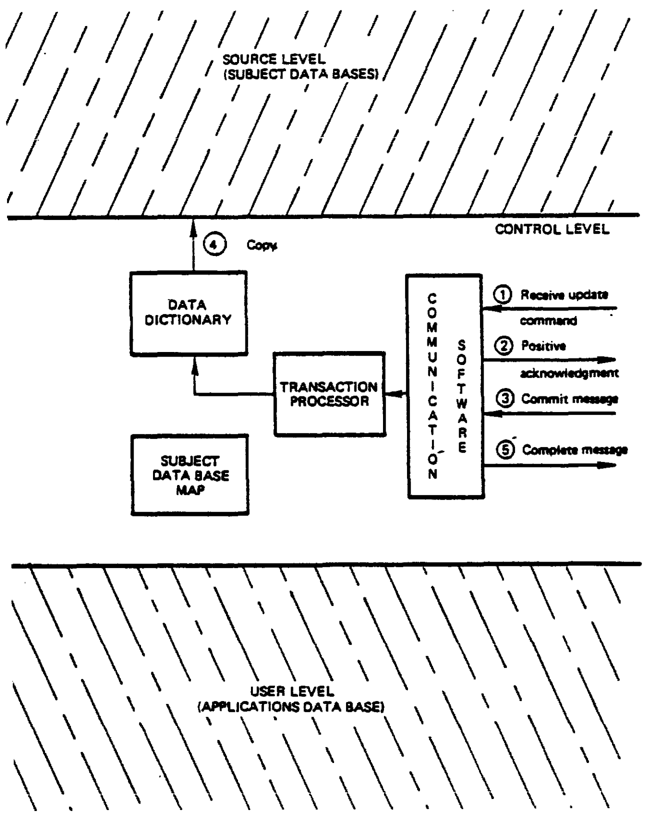


Figure 7—Global synchronization of the source segment

Global synchronization

During normal operation, the source level of the distributed database remains isolated and unchanged. However, the subject databases in the source level must be brought up to date periodically. Figure 7 illustrates how the subject databases in the source level are updated. The system operations center periodically broadcasts an update command to all other regional operations centers. The update command is the first part of a two-phase commit protocol.[5,6] This commit protocol is a data-concurrency protocol that allows global data to be updated simultaneously. When the update command is received, the control segment responds with an acknowledgment that it is ready to update the database. Within a predefined period of time, 5 seconds for example, all the other regional operations centers must also respond to the system operations center with positive acknowledgments to the update command. The commit message, the second part of the protocol, is then broadcast. The copy of the data dictionary is then used to update the subject databases in the source segment.

Any subject database updates received after the first part of the commit protocol is received continue to be stored in the data dictionary until the next update cycle. This avoids conflicts in updating the subject databases when several changes to a given data item occur over a short period of time.

It should now be clear why the broadcast method is chosen for data communications. The broadcast mechanism guarantees that the data updates and the protocol messages arrive at the regional control centers at nearly the same time. In doing so, the subject databases maintain currency and synchronism throughout the network control system.

It has been reported in Reference 7 that the two-phase commit protocol has some drawbacks. That is, if any centers fail to respond to the protocol in either the first or the second phase, the data synchronization process may be kept in a wait state for a long time. However, in this application, since broadcast communication is chosen instead of a virtual ring,[7] the long wait state can be eliminated by broadcasting an abort message after a known elapsed time.

If one regional operations center fails to respond to the two-phase commit protocol, all the source levels of the distributed database will not be updated. Center failure may be temporary or it may extend over a prolonged period of time. A typical temporary condition that normally lasts only a few minutes would be the loss of a satellite communication link because of a severe rainstorm. Such conditions will be overcome at the next periodic global synchronization. However, after several tries and failures (the number is a system tuning parameter determined during operation), some network operations decisions will be made. Since the network control system has backup capability, one of the other regional operations center may be directed to assume the responsibility of controlling the geographical region that has repeatedly failed to respond to the commit protocol. On the other hand, failure may be caused by problems other than computer failure. In this case, the failing center may be excluded by the system operations center from the distributed network (manual entry at operator's console) and may be permitted to operate as an independent subnet within the network. Global synchronization, then, can resume among all other nodes. The next update/commit attempt will be successful within the remaining network. In either event, when the excluded center is brought back into operation, the database can be synchronized via the recovery process to be discussed next.

Note that regional operations are not affected by delays in successfully completing the update to the subject databases in the source level. Operations in a region use the applications databases in the user level, which always have the most current information about the region. Changes in the subject databases that arrive from other centers do not require immediate acceptance since, by definition, they are changes to database items pertinent to those other centers and are required at all centers only to permit backup operations.

It is possible for some delay to occur before the system operations center removes a regional center that is not responding to the commit protocol from the system, in order to permit the remaining sites to complete the subject database update cycle. Should it be necessary for one of the centers to back up the apparently failed center that did not respond, the latest subject database changes still in the data dictionary must be transferred to the source level (and then to the applications level at the backup center) before the start of backup operations. If the update commands are issued by the system operations center frequently enough (the frequency is also a tuning parameter changeable by operator entry in the field), this delay in assuming backup operation will be minimal. Alternatively, the system operations center could issue an update command wherever backup operation is to be initiated as soon as the nonresponding center has been deleted from the active network. This ensures that all available database updates are incorporated into the subject database at the backup center.

Data recovery

Two types of data recovery may occur:

1. Local recovery due to some short-term equipment failures (e.g., power outage for a few seconds) where the subject databases remain intact
2. Remote recovery supported by another center because the original center has been out of service for an extended period of time

Because the user level is oriented toward regional operation and functions independently, local recovery from short-term outage can be treated as a modified initialization process. Using the subject database map, the appropriate subject databases are copied down to the user level and the center becomes operational again. There may be a difference between the database content and the current data values for that region if the center has been inoperative for a period of time and a change in communication asset status occurs during this time. Impacts on network performance due to this time lag have not been investigated in detail. However, it has been reported that a system delay of 10 seconds results in a preblocking probability of 6×10^{-3} for a network control system with assumed customer traffic of 0.1 erlang.[2] (Whenever the called party is busy updating its database image, the telephone call is said to be preblocked.) In any case, for short outages this is not likely to occur, and the status reporting system from the operating units will eventually update the applications database.

After scheduled maintenance or significant (long-term) hardware failure, a specific regional operations center must be brought back up for operation via a remote recovery process. The database of this center will be brought to the current state by that operations center temporarily serving as backup. The subject databases in the source level are slowly updated to the level of last global synchronization. Using the subject database map, which is also received from the backup center, the user level retrieves those needed subject databases from the source level. The center starts functioning but does not start operating online. It is recommended that the center remain hot standby to the backup center until two periodic global synchronizations have occurred. In so doing, the source level is guaranteed to be truly synchronized with the rest of the distributed database and the user level will have the most current information to support regional activities.

IMPLEMENTATION CONSIDERATIONS

Figure 8 shows a computer hardware configuration for a planned implementation of the concept discussed in the previous subsections. This configuration is only one of many that could implement the distributed database design discussed in previous sections. The figure shows that two separate processors are used at each center. The first contains communications interfaces to that region's operating units. These interfaces are computer-to-computer for assets that have automated operating facilities and are computer-to-intelligent terminal for the assets with manual operating units. This processor also contains the network control applications programs and peripheral storage for the user-level applications databases. Note that network control software at all regional centers is identical; during processor boot-up only the geographic identification of the center need be entered to initiate operation. The applications database views applicable to that specific region are created automatically by the software. Processor 1 also contains the operator's console for personnel who control the region.

The second computer contains communications interfaces to the system operations center and the other regional operations centers. Updates to subject databases are broadcast to other centers. Update commands from the system operations center are received, acknowledgments to the update are sent back to the system operations center, and commit messages are subsequently received from that center. In addition, status is sent by each regional center to the system operations center

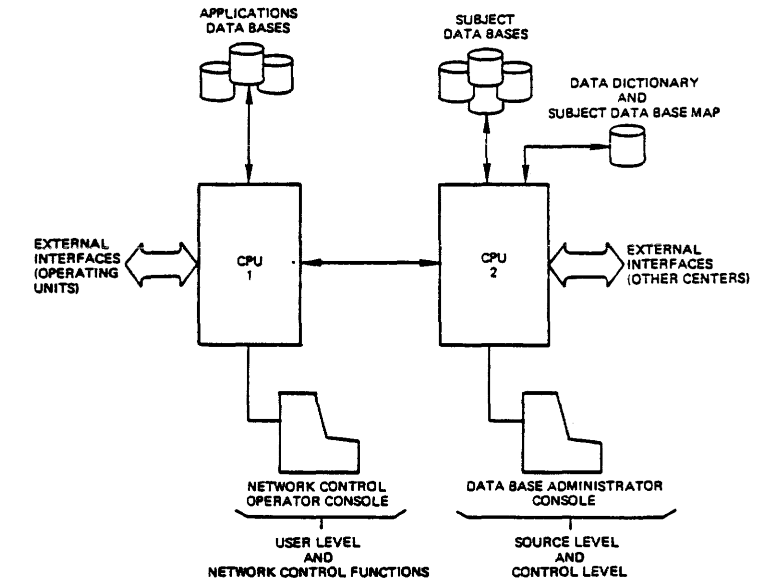

APPLICATIONS DATA BASES
SUBJECT DATA BASES
DATA DICTIONARY AND SUBJECT DATA BASE MAP
EXTERNAL INTERFACES (OPERATING UNITS)
CPU 1
CPU 2
EXTERNAL INTERFACES (OTHER CENTERS)
NETWORK CONTROL OPERATOR CONSOLE
DATA BASE ADMINISTRATOR CONSOLE
USER LEVEL AND NETWORK CONTROL FUNCTIONS
SOURCE LEVEL AND CONTROL LEVEL

Figure 8—Computer configuration for the regional operations center

to permit long-term performance and trend analysis to be performed. These dynamic status data are not part of the subject databases and are transmitted on an infrequent but periodic basis.

The source level (with subject databases) and the control level reside on peripheral storage accessed by this processor. The processor contains the database management software that creates the applications databases pertinent to the specific region. Access to the database management system for manual entry or analysis of the database is possible from the administrator's console, although this does not normally occur. The schema, subschema, and views are created by the database administrator at the system operations center and are distributed as necessary to the regions on disk packs. After system checkout and test, these change only when regional centers are added or deleted, the geographic coverage between centers changes, or communications assets are added or deleted from the network control system. The database administrator, under the direction of the system operations center operator, can update the applications databases in processor 1 to include additional views that cover data for a region requiring backup control.

If central processing unit (CPU) 1 fails, another regional center will take over operation and control of the affected operating units. If CPU 2 remains operational, subject database updates from the other regional centers will continue to be processed (including those from the center acting in a backup capacity). When CPU 1 and its peripherals are restored to operation, a local recovery as previously described can occur.

If CPU 2 or a critical peripheral fails, the system operations center could elect to permit the regional center to continue controlling its area using CPU 1. It would, however, be isolated from the rest of the network and would neither issue nor receive subject database updates. Alternatively, the system operations center could request that another regional center take control. In this case, remote recovery would occur when the faulted equipment is restored to operation.

Figure 9 shows a computer configuration for the system operations center. Only one system center will be implemented; however, one regional center will be provided with sufficient dial network interfaces so that it can act as a backup to the system operations center. Two processors are used at the system center. The first contains network control applications programs similar to those at the regional centers. The user-level applications databases consist of those assets that cross region boundaries. The applications databases are created from views of the subject databases in the source level in a manner similar to that employed at the regional centers. No operating units report directly to this center. Network control information is collected from operating units by the cognizant regional center, and if an interregional link is involved, these data are passed to the system operations center as part of a normal subject database update cycle.

Unlike the regional centers, the system operations center relies entirely on subject database updates (as opposed to real-time inputs from the operating units) for change information. Therefore, the applications databases associated with processor 1 are periodically refreshed from the subject databases (also a tuning parameter). A lag could occur between the occurrence of a database change that affects an interregional link and the refreshing of the applications database. The system operator will tolerate any such potential discrepancies since anomalies that affect interregional links are infrequent (compared to the frequency of intraregional faults), and the delay between the occurrence of a change and its propagation into the applications databases at the system operations center is small (minutes). Processor 1 also contains software,

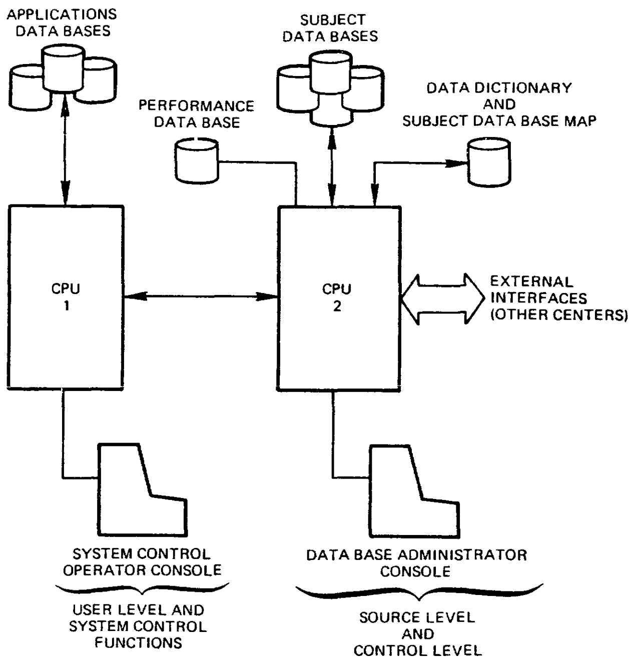

APPLICATIONS DATA BASES
SUBJECT DATA BASES
PERFORMANCE DATA BASE
DATA DICTIONARY AND SUBJECT DATA BASE MAP
CPU 1
CPU 2
EXTERNAL INTERFACES (OTHER CENTERS)
SYSTEM CONTROL OPERATOR CONSOLE
DATA BASE ADMINISTRATOR CONSOLE
USER LEVEL AND SYSTEM CONTROL FUNCTIONS
SOURCE LEVEL AND CONTROL LEVEL

Figure 9—Computer configuration for the system operations center

activated only at the system operations center, that permits the operator to initiate subject database updates throughout the system via the commit protocol. This software exists at all centers (since one regional center will be designated as a backup to the systems center); however, it is loaded only at the center that has been initialized as the systems operation center. The operator also decides whether one regional center must back up another or whether a region is to operate independently. Implementation of backup operation is coordinated manually between the system operator and the regional center database administrator using voice orderwire channels (i.e., it is not automatically implemented via software).

The second computer in Figure 9 contains communications interfaces to the regional operations centers. Updates to subject databases are received (the system operations center does not originate changes to any subject database). Dynamic status data is also received on an asynchronous but periodic basis for long-term performance analysis. The communications software also broadcasts the database update command, receives acknowledgments from the regional centers, and subsequently issues the commit command. The subject databases in the source level (identical to those at the regional centers) and in the control level reside on peripheral storage accessed by this processor. In addition, a series of performance and trend analysis programs can be exercised by the database administrator in a background, low-priority mode. These programs recover selected status data reported by the regional centers from history files in the performance database. The analysis is performed to permit management to assess how well the network control system is performing and to support long-term planning. A number of additional considerations related to system realization are now discussed.

Cost

Costs of hardware will increase because additional processing capability and mass storage capacity are required at each center. However, the number of control centers is relatively small, the total size of the subject databases is moderate, and the cost of processing and peripheral equipment is continually decreasing. Further, the three-level approach localizes the management functions of distributed databases to a particular level—the control level. This permits currently available database management systems to be used in the source and the user levels. This flexibility and use of existing vendor software will offset some or all of the increased system cost.

Data Dictionary

Although the data dictionary has far more promising capabilities, it has traditionally been used as a documentation tool. Currently, work is progressing on expanded data dictionaries, as required by this application.[11] To implement the proposed distributed database system design, the development of enhanced data dictionaries compatible with a variety of vendor database management systems is desirable. Some of the implementation considerations, such as cost tradeoffs and data dictionary development, need further examination and evaluation during project implementation.[12]

CONCLUSION

In summary, a three-level distributed database concept is proposed to resolve the constraints imposed on fully replicated distributed databases. Functions of the three levels (source, control, and user) are discussed. Interactions among the three levels are also described by considering four operational stages: initialization, continuous update, global synchronization, and data recovery.

Data are globally synchronized at the control level. To do this a broader usage of the data dictionary is proposed. Besides being used as a documentation tool, the data dictionary is also used as a bookkeeper to maintain records of global data updates.

The distributed database concept discussed here is applicable to dissimilar, but analogous, applications. For example, consider a multinational corporation. Each division of the corporation resembles the regional operations centers. Local inventory can be updated locally from minute to minute while the same set of data is distributed and kept current throughout the corporation for other non-real-time processing.

The database system design proposed in this paper allows currently available database management systems to be used in the source and user levels. Development of the components of the control level is necessary, and some progress has already been reported. By introducing the three-level concept, the impacts on network control operations due to requirements for database concurrency are minimized.

ACKNOWLEDGMENTS

The authors wish to thank Carl Hellman, Edward Hirshfield, and Dr. Michael Sites of Ford Aerospace & Communications Corporation for their encouragement and continued support during the course of this work.

REFERENCES

1. Sanli, N., S. C. Lo, S. L. Kota, and M. H. Aronson. "The NATO III Satellite Communications System Control." *Proceedings of the AIAA 9th Communications Satellite Systems Conference,* San Diego, Calif., March 7–11, 1982, pp. 255–262.
2. Mase, K., M. Kajiwara, H. Yamamoto, and M. Shinohara. "Network Control System Using Traffic Databases." *IEEE Proceedings of the International Communications Conference,* Philadelphia, June 13–17, 1982, pp. 1F.1.1—1.F.1.5.
3. Mohan, C. "Distributed Data Base Management: Some Thoughts and Analysis." *Proceedings ACM 1980, Annual Conference,* Nashville, TN, October 1980.
4. Ramamoorthy, C. V., S. T. Dong, S. L. Ganesh, C. H. Jen, and W. T. Tsai. "Architectural Issues in Distributed Systems." *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management,* Hot Springs, Va., November 11–13, 1981, pp. 1–11.
5. Garcia-Molina, H., and D. Barbara. "The Cost of Data Replication." *Proceedings of the 7th Data Communications Symposium,* Mexico City, October 27–29, 1981, pp. 193–198.
6. Garcia-Molina, H. "Reliability Issues for Fully Replicated Distributed Databases." *IEEE Computer Magazine,* September (1982), pp. 34–42.
7. Le Lann, G. "A Distributed System for Real-Time Transaction Processing." *IEEE Computer Magazine,* February (1981), pp. 43–48.
8. Mohan, C. "Some Notes on Multi-Level Data Base Design." Technical Report TR-128, Department of Computer Sciences, University of Texas, Austin, TX, May 1979.
9. Gerla, M. "Routing and Flow Control." In Franklin Kuo (ed.), *Protocols and Techniques for Data Communication Networks.* Englewood Cliffs, N.J.: Prentice-Hall, 1981.
10. Curtice, R. M. "DATA DICTIONARIES: An Assessment of Current Practices and Problems." *Proceedings of the 7th International Conference on Vey Large Data Bases,* Cannes, France, September 9–11, 1981, pp. 564—570.
11. Norman, A. "EMPACT (TM)—A Distributed Database Application." *AFIPS, Proceedings of the National Computer Conference* (Vol. 52), 1983.
12. Aronson, M. H., and S. L. Kota. "A Network Control Capability for the NATO III Satellite Communications System Using HP/1000 Computers." HP 1000 International User Group Conference, EXPLOITATION 1983, London, April 1983.

EMPACT™: A distributed database application

by ALAN NORMAN and MARK ANDERTON
TANDEM Computers Incorporated
Cupertino, California

ABSTRACT

EMPACT™, TANDEM Computers' manufacturing information control system, is an application that uses a distributed database. The requirements of the system include supporting multiple sites, providing continuous availability to the data, and controlling updates to communal information. The approach taken to satisfy these needs involves the use of both partitioned and replicated data. Presented in this paper is a discussion of the application requirements, the design and architecture of the database, and the algorithms used for updates and inserts.

INTRODUCTION

EMPACT™, TANDEM Computers' manufacturing information control system, is an application that makes use of a distributed database. The requirements of the system include

1. Supporting multiple sites of independently managed machines.
2. Providing the continuous availability of communal information.
3. Permitting decentralization both of control of the application and management of the data.
4. Allowing for centralized development of the application.

This paper discusses the application requirements, the design and architecture of the database, and the methods used for maintaining the database.

APPLICATION REQUIREMENTS

Application Functions

EMPACT was developed to satisfy the data processing requirements of TANDEM Computer's manufacturing division. It was initially designed as a centralized online application that would support a single manufacturing site. The functions contained within EMPACT pertain to all the standard assembly manufacturing needs. These functions are as follows:

1. Parts master
2. Bills of material
3. Inventory control
4. Purchasing/receiving
5. Work in process
6. Master scheduling
7. Materials requirement planning
8. Interplant material transfer (MART)
9. Job/lot tracking

Database Size and Service Levels

EMPACT was implemented on a TANDEM T/16 computer using TANDEM's distributed data management system, ENCOMPASS. The database size and the required service levels for the initial design were moderate—

1. Single manufacturing site
2. Database of approximately 120 megabytes
3. Database containing 500+ data elements and over 40 record types
4. A user community of 150 persons using 55 terminals
5. Monthly transaction rates: 15,000 against the entire database, with 4,500 update transactions against the bill of material files and 1,000 against the item master file.
6. A 6:1 ratio of reads to writes against the item master and bill of materials files.
7. Hardcopy batch reports generated nightly to supplement the online inquiry capabilities; these reports were created using ENFORM, TANDEM's query language.

Business Environment

TANDEM, as a company, anticipated high growth rates over the next several years. This meant that the size of the user community, the transaction volume, and database requirements were expected to increase accordingly. Furthermore, the manufacturing management decided to decentralize its operations. The plan was and is to disperse the manufacturing effort into several plants, all performing similar operational functions although geographically distant. A growth rate of one new plant every 9 months was predicted.

This need to support geographically distributed sites placed some new demands upon EMPACT. As a business requirement, communal data such as item master or bill of materials information would have to be continuously available; the day-to-day business of the individual sites is highly dependent on this information. Also, owing to the decentralized management structure of TANDEM's manufacturing organization, the sites required as much autonomy as possible; the daily business of one site could not adversely affect or be affected by events or circumstances at other sites. Each site was also to be responsible for and have control over the data stored there.

Initial Development Approach

The conceptual approach taken was to divide the elements of the database into two major categories, global data and local data, and one additional category that must be acknowledged but does not enter into this analysis, private data—

1. Global data are information that would be common to and shared by all sites. Examples of global data are the list of parts that determine TANDEM's parts catalog (item master file) or the records that describe TANDEM's product structure (bill of materials file).
2. Local data are information that is uniquely important to the individual site using it but accessible by all sites.

Examples of local data are stock status and work-in-process data. Local data have the same format as the corresponding data at other sites.

3. Private data are information that is only used by the individual site and is not visible to other sites. This data consists of reports and data outside the EMPACT system.

Global and local data are each segregated into record units that are themselves either global or local. This segregation is not a requirement of the distributed application but a means of simplifying certain aspects of the application.

Figure 1 shows the division of the EMPACT database into global and local portions. As previously stated, the major requirement addressed by distributed EMPACT was site responsibility for the maintenance of local information and providing equal shares of control and responsibility for the maintenance of global information.

Global data are necessarily replicated because of the requirement for site independence. If global data are not replicated at all EMPACT sites, then disconnecting any nonglobal site from the network would mean that business activities at that site would be interrupted until the site rejoined the network. Any scheme of centralized global data suffers from this unacceptable possibility.

By our early reckoning, data would be either global (replicated at all sites) or local (single-site resident). However, another condition surfaced that called for only partial replication of data. The interplant material transfer function, known by the acronym MART, permits a request by one site for material from another to be placed and processed, and monitors the subsequent transfer of material. The process requires that all data and status information pertaining to the request be resident at both sites. However, the information is unnecessary to any third party. Such data is classified as semi-global since it only requires partial replication.

The next section provides a brief description of the TANDEM software products that are the foundation of distributed EMPACT.

TANDEM'S SOFTWARE ENVIRONMENT—AN OVERVIEW OF ENCOMPASS-EXPAND

The expansion of EMPACT to support multiple sites was implemented using ENCOMPASS, TANDEM's data man-

```
                          +===============+
                          | Item Master   |
                          +===============+
         +============+                         +===========+
         | Bill of    |                         | Inventory |
         | Materials  |                         |           |
         +============+    ************         +===========+
                          *    Data    *
     +=============+     *  Dictionary  *          +===========+
     | Purchasing  |    *                *         | Work in   |
     | Receiving   |     *   Programs   *          | Process   |
     +=============+      *            *           +===========+
                           ************
         +=============+                        +============+
         | Master      |                        | Job/Lot    |
         | Schedule    |                        | Tracking   |
         +=============+                        +============+
                           +=============+
                           |    MART     |
                           +=============+
```

Global Data	Local Data	Semi-Global Data
Bill of Materials	Inventory	MART
Item Master	Master Schedule	
Global data dictionary	Purchasing	
Global programs	Work in Process	
	Job/Lot Tracking	

Figure 1—Structure of the EMPACT database

agement system, and EXPAND, TANDEM's network software. ENCOMPASS is a conglomerate of several major software products, included in which are ENSCRIBE, DDL, ENFORM, PATHWAY, and TMF, the transaction and recovery manager.

ENSCRIBE is a relational-database manager that supports three types of structured-file organizations: key-sequenced, relative, and entry-sequenced. Within these file types, data are organized into logical records. EMPACT restricts itself to fixed-length records and key-sequenced files.

DDL is a data definition language used to define the data elements and record structures that make up an application. The output generated by DDL is stored in a data dictionary that is used by ENFORM, a high-level nonprocedural query language, to generate reports against the application database.

PATHWAY provides a terminal-oriented interface for transaction design and control. Its primary components are a terminal control process (TCP) that provides screen formatting and control, user-written programs, known as *servers,* that access and update the data files using ENSCRIBE procedures, and PATHMON, a monitor process that dynamically manages and allocates application resources.

An application's interface to the end-user is a set of program modules, or *requestors,* that are written in Screen-COBOL (a COBOL-like language with extensions for screen manipulation). The Screen-COBOL programs are interpreted by the TCP to perform screen sequencing, data mapping, and transaction control. The Screen-COBOL programs communicate with the servers by exchanging request and reply messages, hence the names requestor and server.

The servers are single-threaded programs that

1. Read a request message
2. Perform the database function requested
3. Reply

They must be *context-free,* meaning that they do not need a memory of past requests in order to execute the current request.

A server is known to the requestor programs by its *server class name.* A *server class* is a group of server programs that perform identical functions. When a TCP needs to establish communication with a particular server class, it requests a 'link' to an individual server within the given server class from PATHMON (see Figure 2). PATHMON responds by sending the TCP the process name of an already existing server or of a newly created one. How PATHMON decides to respond depends on parameters established when the system is started up. This is one aspect of the load balancing and resource management that PATHMON performs.

Transactions in ENCOMPASS are defined as multistep operations that change the database from one consistent state to another. They are processed via the requestor/server relationship. The end user initiates a transaction request at a terminal; the requestor module responds by sending a series of one or more requests to a set of servers; the servers perform the application function against the database; replies are re-

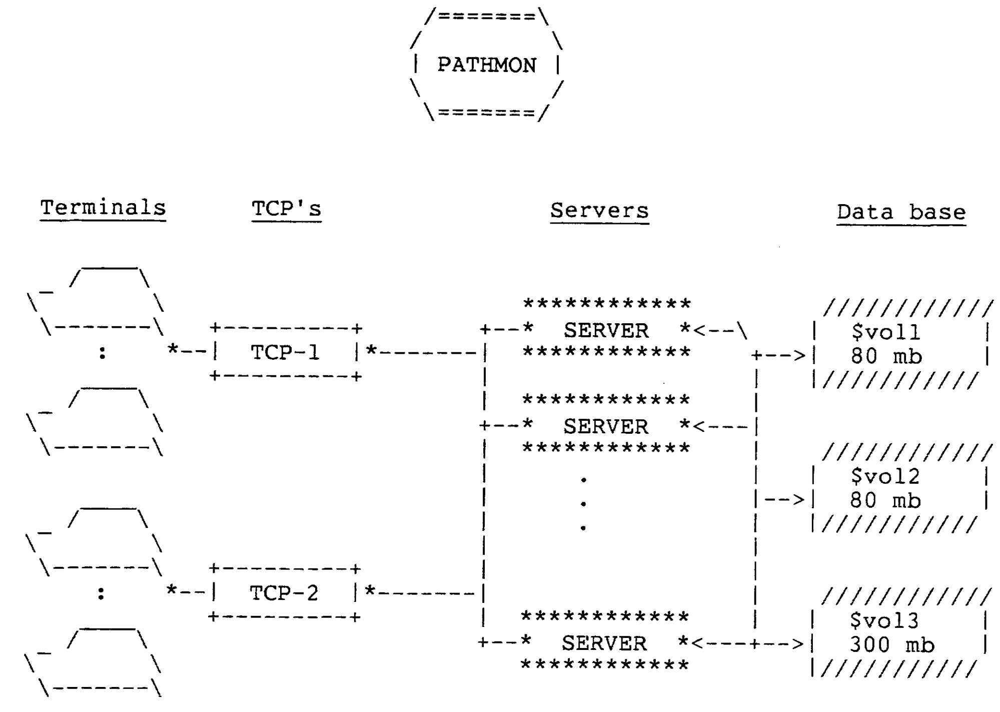


Figure 2—Pathway configuration for EMPACT at a single site

turned to the requestor; and the end user is notified of the result at the terminal. Smith discusses the requestor/server relationship in more detail.[4]

TMF guarantees the consistency of the database by ensuring that precisely all or none of the changes that a transaction attempts to make are permanent. For a more detailed discussion on TMF, refer to Borr's report.[1]

EXPAND, TANDEM's network software, has several features relevant to application design. Control is decentralized; it is characterized by the lack of a network master. Access to geographically distributed system resources is transparent to the system user. In fact, local and remote requests to servers or files look the same to those programs initiating the requests. Nodes in EXPAND network are defined by their system name, for example \LA or \SANFRAN. Interprocess communication across the network requires only that the system name be specified, all other aspects being managed by the operating system. The design of the EXPAND network is described in more detail by Katzman and Taylor.[2]

ENVIRONMENT—EMPACT

EMPACT PATHWAY Configuration

EMPACT, prior to becoming distributed, had over 120 requestors and 40 server classes. There were over 200 transaction types and approximately 100 different reports. Typically, each subsystem, for example, bill of materials, would have three server classes associated with it: one for updates, one for online queries, and one for report generation. Figure 2 shows the EMPACT PATHWAY configuration.

Application Development

The program development environment consisted of a centralized team of programmer/analysts working with management and manufacturing personnel throughout the company. It was agreed that the development of software that could modify the EMPACT database should be controlled by the central team in order to prevent inconsistent alterations to the database. However, all other software, for example, for reports, could be developed anywhere and fall under the control of the intended end user. All released software is archived in a central and controlled repository; this decision aimed to protect the integrity of the released software.

Preliminary Designs

At the time the design for EMPACT's distribution was being considered, TANDEM had a 50-site corporate network. EMPACT is just one client of this corporate network, which is also used for electronic mail, text processing, program development, and many other EDP applications at TANDEM. The network has now grown to 150 sites and is expected to grow accordingly. Each manufacturing site was to have its own system. The question that faced EMPACT's designers was how best to utilize this network to support multiple sites.

The available options were:

1. Remain centralized, have one central database under the control of a single PATHWAY system with remote users accessing the system over the network.
2. Partially distribute the database, have a small number of geographically dispersed *master* sites running independent but identical versions of EMPACT that supported users on sites within a defined region.
3. Fully distribute the database and application: have complete copies of the database and software running at each site.

The first option violated the requirement for continuous availability of the database. Despite TANDEM NonStop™ system design, long-haul communications media are still subject to occasional interruption.

The second option was a compromise solution; vulnerability to significant communications interruptions might, arguably, be reduced, but the principle of site autonomy would not be satisfied. Some individual site or sites would still be dependent on a particular master site. Furthermore, given the geographic dispersion of manufacturing sites, it was unclear whether sites would ever congregate in sufficient numbers to afford nonmaster sites.

These considerations led to the decision to select the third option as the basis for implementing distributed EMPACT. The next section of this paper describes the specific approach taken.

DISTRIBUTION APPROACH

Data Qualification

The primary components of the design of distributed EMPACT are (see Figure 3)

1. Each site has an independent PATHWAY system running identical applications.
2. The database structure for global and local data—as opposed to the data themselves—is global, hence identical at each site.
3. The global portion of the database is replicated at each site, that is, both the structure and the data of files with global information are identical at all sites.
4. The local portion of the database is partitioned over the network, that is although the structure of files containing local information is identical at all sites, Austin stores the Texas data and Neufahrn stores the Germany data.

Database Consistency Solutions

This design satisfies some of the objectives of continuous availability and site autonomy. Because a user is concerned only with his/her portion of the local data and because the global portion is replicated everywhere, query access to the

```
+=============+                                   +=============+
|             |                                   |             |
| SANTA CLARA | --------------------------------- |   RESTON    |
|             |\                                 /|             |
+=============+ \                               / +=============+
      |          \                             /         |
      |      \\\\\\\\\\\\\           \\\\\\\\\\\\\       |
      |      \  Local    \           \  Local    \       |
      |      \   Data    \           \   Data    \       |
      |      \\\\\\\\\\\\\           \\\\\\\\\\\\\       |
      |      \  Global   \           \  Global   \       |
      |      \   Data    \           \   Data    \       |
      |      \\\\\\\\\\\\\   \   /   \\\\\\\\\\\\\       |
      |                       \/                         |
      |                       /\                         |
      |      \\\\\\\\\\\\\   /   \   \\\\\\\\\\\\\       |
      |      \  Global   \           \  Global   \       |
      |      \   Data    \           \   Data    \       |
      |      \\\\\\\\\\\\\           \\\\\\\\\\\\\       |
      |      \  Local    \           \  Local    \       |
      |      \   Data    \           \   Data    \       |
      |      \\\\\\\\\\\\\           \\\\\\\\\\\\\       |
      |          /                             \         |
      |         /                               \        |
+=============+ /                                 \+=============+
|             |                                   |             |
|   AUSTIN    | --------------------------------- |  NEUFAHRN   |
|             |                                   |             |
+=============+                                   +=============+
```

Figure 3—Example of EMPACT network and database structure

database is guaranteed regardless of the status of other sites in the network or the status of the network itself. However, the issue remains how to provide similar guarantees for update access to the global data yet still maintain consistency among the files.

Maintaining the consistency of replicated files requires that any changes to an individual copy be made to the corresponding copies elsewhere in the network. Furthermore, the changes must preserve the consistency of the files in the event of failure.

Preliminary Broadcast Transaction

One method proposed was to modify the existing servers updating global files to *broadcast* the updates to all the sites in the network as a single transaction. The transaction would be performed under the auspices of TMF, hence there would be a guarantee, through the BACKOUT and/or ROLLFORWARD mechanisms of TMF, that the consistency of the database would be maintained in the event of failure.[1]

This proposal, however, has three drawbacks: servers that perform updates against global files require knowledge of the whereabouts of the file replicas; there is a long response time at the user terminal during an update to global data; and updates to the global files can only take place when all sites are available.

Owing to the fact that the EMPACT sites have a variety of hardware and system configurations, the naming of global files (system name, volume, subvolume) is unique to each site. Because it accesses them directly, the server that performs updates to global files needs to know the names of all global-file copies to be updated. Sites in the network have to keep current with the naming changes of all other sites—a problem that becomes considerably worse as the number of EMPACT sites increases.

The terminal response time is unacceptably long because an update to global data involves updating a larger number of file copies. On the basis of operator wait-time, daily business requests cannot rely exclusively on this method of update propagation.

The availability of all sites is necessary because by design the broadcast transaction requires access to all affected files at the same time. Data integrity demands an all-or-nothing update. Entrusting the daily business of a site to this method alone sacrifices site autonomy, since legitimate global activity would have to be interrupted until communications were regained. Figure 4 shows the structure of the preliminary broadcast transaction.

Broadcast Transaction

A refinement to this proposal defines servers at each site, known as *associate* servers, that upon receiving a request from an updating server at an initiating site, perform the updates against their copies of the global files. Furthermore, a new global file, known as the node file, contains a list of all EMPACT nodes in the network and thus serves as a network map. A broadcast transaction now involves sending a request to servers at each of the sites listed in the node file. The servers at remote nodes have knowledge of all physical file placement

```
      \STCL

 +------------+
 | REQUESTOR  |
 +------------+                  ///////////
     |   ^                    -->| \NEUFAHRN |
     |   |                       | DATABASE  |
 (1) |   | (6)        (3) /      ///////////
     |   |               /
     v   |              /        ///////////
 +------------+        / (4)     | \RESTON   |
 |   SERVER   | ------*----->    | DATABASE  |
 +------------+        \         ///////////
   |                    \
(2)|              (5)    \       ///////////
   v                      \   -->| \AUSTIN   |
///////////                      | DATABASE  |
| EMPACT    |                    ///////////
| DATABASE  |
//////////
```

Figure 4—Preliminary broadcast transaction: (1) Requestor sends transaction-request message to server; (2) server updates local copy of database; (3)–(5) server updates all known remote databases; (6) server replies to requestor.

at their respective sites. TMF guarantees the consistency of the database by covering each broadcast. Figure 5 presents the structure of a broadcast transaction with node file and remote server.

This method of broadcasting transactions resolves the problem of naming, but it still violates the requirements of site autonomy and short terminal response time. A transaction that updates global data can only be performed when all sites in the network are available. Furthermore, as the number of sites in the network becomes large, the response time on the terminal becomes unreasonably long.

Suspense Transaction

The selected solution to both of these problems was to sacrifice the absolute consistency of the replicated files in exchange for site autonomy and short terminal response times by using a *suspense* mechanism to maintain database consistency.

Instead of immediately broadcasting the transaction to all sites in the network, the server at the site where a global transaction is initiated first updates its copy of the global file and then posts the transaction message to a suspense, or queue, file. A dedicated process, known as the suspense monitor (SUSMON), asynchronously polls the suspense file for transactions and on an as-soon-as-possible (ASAP) basis sends the transaction messages to appropriate servers at remote sites, one at a time, as separate logical TMF transactions. The database is completely consistent only when the suspense files at all the sites are empty. Figure 6 shows the structure of a suspense transaction.

The requirement of site autonomy is satisfied because updates to global files can be initiated regardless of the status of other sites in the network. The problem of unsatisfactory response time at the user's terminal is resolved because the server at the initiating site has only to update the local database before sending a reply back to the TCP. The propagation of the update to remote sites is performed asynchronously by SUSMON. All update and insert activity involving the suspense file is covered by TMF.

Data Ownership and Key Ranges

As with the broadcast method, TMF guarantees the consistency of the database. However, with TMF, unlike the broadcast method, because the suspense mechanism introduces a time delay in the propagation of updates to remote sites, the

```
      \STCL
 +------------+
 | REQUESTOR  |
 +------------+                                          ///////////
     |   ^                        +----------+ (5)  | \NEUFAHRN |
     |   |                    --> | \NEUFAHRN| -->  | DATABASE  |
 (1) |   | (10)             /     | SERVER   |      ///////////
     |   |             (4) /      +----------+
     v   |                /                          ///////////
 +------------+          / (6)    +----------+ (7)  | \RESTON   |
 |   SERVER   |<-------*----->    | \RESTON  | -->  | DATABASE  |
 +------------+         \         | SERVER   |      ///////////
     |          ^        \        +----------+
 (2) |          | (3) (8) \                          ///////////
     v          |          \      +----------+ (9)  | \AUSTIN   |
///////////  +------+        --> | \AUSTIN  | -->  | DATABASE  |
| EMPACT    | | NODE |            | SERVER   |      ///////////
| DATABASE  | | FILE |            +----------+
//////////   +------+
```

Figure 5—Broadcast transaction: (1) Requestor sends request message to server; (2) server updates local copy of database; (3) server reads node file to determine where else to send transaction; (4), (6), (8) server sends request to associate servers at remote sites; (5), (7), (9) associate servers perform update; (10) servers reply to requestor.

```
      \STCL

   +------------+
   | REQUESTOR  |                                           (7)  //////////
   +------------+                              +----------+      |\NEUFAHRN|
        |  ^                                   | \NEUFAHRN| -->|DATABASE |
        |  |                               --> | SERVER   |    |//////////
    (1) |  | (4)                           /   +----------+
        |  |                           (6) /                (9)   /////////
        v  |                              /    +----------+       |\RESTON |
   +------------+           +--------+ / (8)   | \RESTON  | -->  |DATABASE|
   |   SERVER   |       /-->| SUSMON |*----- > | SERVER   |      |/////////
   +------------+      /    +--------+ \       +----------+
       |      |       /  (5)            \               (11)  /////////
   (2) |      | (3)  /              (10)\    +----------+     |\AUSTIN |
       v      v     /                    \   | \AUSTIN  | --> |DATABASE|
 ///////////  +----------+               --> | SERVER   |     |/////////
 | EMPACT    | | SUSPENSE |                   +----------+
 | DATABASE  | |   FILE   |
 |////////// +----------+
```

Figure 6—Suspense transaction: (1) Requestor sends request message to server; (2) server updates local copy of database; (3) server posts message in suspense file; (4) server replies to requestor; (5), (7), (9) SUSMON asynchronously sends transaction to remote servers; (6), (8) (10) remote server attempts update at remote database and responds to SUSMON.

possibility of *conflicting* adds and updates among the sites becomes a problem.

Conflicting updates occur when two or more sites update their copies of the same data simultaneously. Similarly, conflicting adds occur when two or more sites add records to a global file with the same key value, but different data. For example, a conflicting add would occur if Reston and Austin were to simultaneously add the same part number to their copies of the item-master file, but have different physical items associated with that number.

To prevent this possibility, ownership (by site) is assigned to global records and the initiation of updates is restricted to only the owning site. To prevent conflicting adds, the ranges of key values are preassigned to the various sites and adds are limited to those ranges.

Although only one site at a time can have add or update rights to a particular record or range of records, these rights could be given away to another site. The user community did not find this an unreasonable limitation.

Stale Data

Another problem introduced by the suspense mechanism is the problem of *stale* data. Stale data occur when an out-of-date copy of a global file is read. The data are out-of-date because an update to the file has been posted at a remote site but has not yet been propagated to the local site. However, because the propagation time for suspense updates is considerably less than the time the user community takes to act on the update, temporary staleness is not a problem.

Before the implementation of distributed EMPACT, the various manufacturing sites were using shared information that was updated and reconciled on a weekly basis. In distributed EMPACT the elapsed time for the propagation of a transaction to all sites in the network is less than a minute; in the worst case, such as a catastrophic network line failure, it is several days. This is viewed as a major improvement. It was also pointed out that if indeed the most current information were absolutely needed, the read could be directed to the copy at the owner site. If a business emergency were to occur that could not wait on the suspense delay, an outside mechanism such as a phone could be used.

In short, the approach taken to satisfy the simultaneous requirements of site autonomy and database consistency is to replicate communcal information, use an ASAP update scheme to maintain the concurrency of the replicated information, and rely on the notion of record ownership to prevent conflicting updates. The favorable relation of propagation time to action time permits brief periods of stale data.

The next section of this paper discusses some of the details related to the implementation of this approach.

IMPLEMENTATION

The conversion of EMPACT from a centralized application to a distributed application required

1. Specific modifications to the structure of the database
2. Developing the suspense-processing utility SUSMON
3. Identifying data consistency algorithms
4. Developing a means to serialize transactions
5. Deciding upon standard update and record-locking protocols
6. Handling processing variations and exceptions
7. Error processing

Database Restructuring

The existing database was reorganized so that the local, global, and semiglobal information reside in distinct sets of files that themselves are solely global, local, or semiglobal. As was mentioned earlier, this restructuring simplifies the organization and reduces the complexity of the update servers. The logic that updates global information can be separated from the relatively simpler logic that updates local information, easing the task of program conversion and maintenance. This conversion is necessary for all programs that access global data.

Suspense File Monitor

The SUSMON process at each site, as shown in Figure 6, is designed to propagate any transaction posted in its suspense file to other sites in the network. Each site has its own suspense file and copy of SUSMON. SUSMON, in conjunction with the servers that update global data, maintains the consistency of replicated data.

SUSMON's map of the network is the node file, which is replicated at all the EMPACT sites. This file contains the Guardian system names and statuses of all sites.

Records in the suspense file contain a copy of the request message, the name of the server class that receives and acts on the message, and a *bit mask* that is matched against the list of sites in the node file to identify where the transaction has been sent or needs to be sent.

SUSMON polls the suspense file looking for work. When finding a transaction that needs to be sent to a remote site, SUSMON performs the following actions:

1. Invokes TMF monitoring—initiates the transaction.
2. Establishes communication with the appropriate server class at the remote site.
3. Sends the request message to the server.
4. Waits for a reply.
5. If the reply is affirmative, updates the bit mask to indicate that the site has processed the transaction successfully.
6. If some site is not available, either because a network link was down or because a system is unavailable, the algorithm defers, intending to try again at a later time.
7. When the bit mask indicates that all necessary sites have received the transaction, the process deletes the record from the suspense file.
8. Ends TMF monitoring—commits the transaction.

Figure 7 diagrams the information flow during suspense processing.

Data Consistency Algorithms

Given the restructured database and the development of SUSMON, the conversion task was directed towards modifying those servers that update global files. The design of a new update algorithm was required to resolve concurrency problems and to provide a guarantee of data consistency within the replicated files. The original update algorithm, a protocol that was designed for a single-site application, plays an extended role in the distributed scheme; we will now describe it before moving on to its modifications.

A standard two-step protocol of *check* and *update* is used for update transactions. As part of the first step, an end user

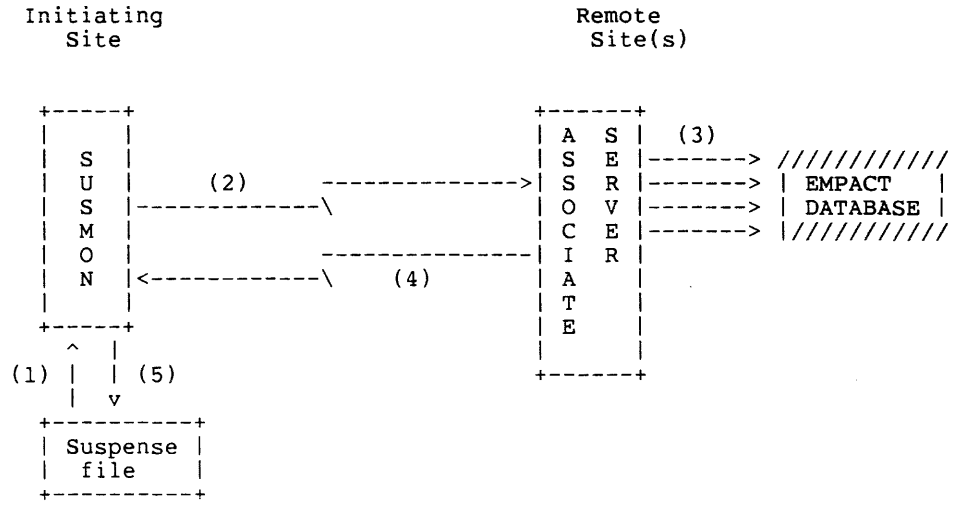


Figure 7—Suspense processing: (1) SUSMON obtains transaction from suspense file; (2) SUSMON establishes communication with server at remote site and sends request message; (3) server updates local copy of database—may involve multiple records; (4) server replies to SUSMON; (5) SUSMON updates bit mask of record in suspense file.

```
+----------------------------+
|                            |
|     R E Q U E S T O R      |
|                            |
+----------------------------+
     |   ^         |    ^
 (1) |   | (3) (4) |    | (7)
     v   |         v    |
  +----------------------+
  |    S E R V E R (S)   |
  +----------------------+
      ^ ^       ^ ^   ^ ^
  (2) | |   (5) | |   | | (6)
      v v       v v   v v
   /////////////////////
   | EMPACT DATA BASE  |
   |////////////////////
```

Figure 8—Check and update protocol: (1) Requestor sends check message; (2) Server accesses database and verifies validity of transaction; (3) server replies; (4) requestor initiates update step; (5) server performs validity checks; (6) server updates database; (7) reply is returned to requestor. Note that each access to the database—steps (2), (5), and (6)—may be multiple.

might access a screen and make a request to update a certain record or piece of information in order to add a component part to an existing bill of material structure. The requestor module servicing the end user then sends the appropriate server class a request for the record or piece of information specified. Next, the server verifies that such an update is possible or permitted, and it replies with the requested data. (See Figure 8.)

To add a new component, the server needs to verify, among other things, that the assembly exists, that the component to be added is an existing part, and that the assembly is not a component of the component—bills of materials cannot be recursive.

After the user is satisfied with the data entry, the requestor module initiates the second phase of the update process: a request to a server (not necessarily the process that handled the first request) asks that the update be performed against the database. The updating server again verifies that the update is possible, repeating the validity checks that were used in the first step. If all validation is affirmative, the database is updated, a successful reply is sent to the requestor, and ultimately to the end user. In the event of error, a negative reply is returned.

The purpose of this two-step check and update protocol is to provide a reasonable level of user friendliness and concurrency. The check allows the user to find out prior to performing potentially unnecessary data entry whether or not a given change is actually needed and possible. A higher level of concurrent processing is achieved because data locks, which are handled at the record level, are only held while the actual update is being performed by the server, and not while data are being entered.

Conflicting updates within a site are prevented because an *old-data/new-data* comparison is performed before update. If the current copy of the data in the database matches the old copy of the data in the update request message, then the new updated data are written to the database. Otherwise the update request is rejected because a legitimate update has taken place between the time the check request was issued and the time the update request was made.

This two-step method for performing updates still applies within the environment of suspense processing. The check portion of the transaction remains unchanged, but the update portion requires additional logic to make use of SUSMON.

Upon successful completion of an update, instead of simply replying affirmatively to the requestor, the updating server first writes a copy of the update request message to the suspense file, then replies. SUSMON asynchronously sends the message, as posted, to identical servers at each of the remote sites. These servers perform the same processing logic that is performed at the initiating site, including validity checks; however, no copy of the message is posted in the local suspense file, since the destination of the update has been reached.

Using identical versions of the update server to process a given request at both the initiating site and the receiving nodes minimizes program maintenance; there is no need for multiple servers with nearly identical logic. Moreover, the consistency of the replicated copies of the data is continually being verified. If a replica becomes inconsistent, this mechanism will detect the circumstance.

This second feature, however, places several additional constraints on the structure of updates to global data. SUSMON sends requests to remote sites serially, that is, requests are sent to remote nodes one at a time and only after the previous one has successfully completed. Because the validity checks are also being performed at remote sites, the success of a global request at a remote node is guaranteed only if the transactions processed at the initiating site are serializable.

Transaction Serialization

A set of transactions is considered serializable even though it is processed in an interleaved or concurrent fashion, if there exists some serial order that will produce equivalent results.

The method chosen to accomplish this ordering at a node initiating global updates is a counter called the SUSMON-TRANS-ID. The counter is a 32-bit field residing on a single record in the database. Initially the counter is set to zero. Servers processing global transactions are required to read, lock, and increment the counter prior to posting the request message in the suspense file but subsequent to the completion of all standard processing.

Serialization is achieved using the SUSMON-TRANS-ID if

1. All records read and written are locked from the time of access to the time that the transaction is committed.
2. All records written (added, updated, or deleted) are locked prior to the write operation.
3. All records read are global.

This last clause simply implies that there can be no dependencies on nonglobal information, because the condition of non-

```
^                                       *
|                                 *     .
|                            *
|                       *               .
|                  *
|             *                         .
|        *
|   *                                   .
+---------------------------------------*--->  Time
                                        |
 \ /\________________________/\________/ |
(1)            (2)               (3)    (4)
```

```
(1)  TMF transaction initiated.

(2)  Reads and writes performed for standard processing,
     all locks held.

(3)  Obtain SUSMON-TRANS-ID and post message in suspense file.
     (Only one transaction per site is in this step at one time)

(4)  Commit transaction and release locks.
```

Figure 9—Resource acquisition as a function of time during a global transaction

global data, by definition, may vary from site to node. Figure 9 charts the resource acquisition required for serialization.

The suspense file is key sequenced. The value obtained by incrementing the SUSMON-TRANS-ID counter determines the relative position of the record in the suspense file; this value is used as the primary record key. The order derived is the order in which the transactions will be propagated to other nodes.

Record-Locking Protocol

The first step of the locking protocol requires that all global records that are read also be locked; this ensures data consistency through the duration of database accesses. The second step of the locking protocol requires that the SUSMON-TRANS-ID be locked by any server process that uses the suspense file; this guarantees serialization of outbound transactions. The TMF Transid, however unique, is not capable of being used to serialize transactions; it is acquired at the time that the transaction is initiated, and not at the time that the suspense file is written to. Figure 9 clarifies this point.

Variations and Exceptions

The majority of the transactions in EMPACT easily adhere to this structure. There are three notable exceptions, however: the deletion of a part from the item-master, the addition of new global records, and the interplant transfer of material.

Part information in EMPACT resides in the item master record, which is defined to be global. Deleting an individual part from the database involves deleting the associated record from the item master file, but only after inactivity for the part has been established. A part is considered active if inventory records show a stock quantity for the item, or if the part is called out by any assemblies in the bill of materials records. In short, the activity of a part is defined by both global and local information, and it is this dependence on local information that undoes the algorithm outlined above.

The resolution to this problem was the implementation of part deletion as a broadcast transaction. Because the frequency of this transaction is historically low and because the day-to-day business of the individual sites does not depend on their being able to perform this transaction, the fundamental vulnerabilities of broadcast transactions are acceptable. The lesson learned by the designers of EMPACT was that if a particular transaction did not fit the structure required for suspense processing, broadcasting could serve to perform it.

The decision to assign ranges of key values in order to prevent conflicting adds was also the result of an informative debate. Replication requires a common recognition of a record's global uniqueness; for example, a part number can only identify one kind of part throughout the system. If any site were permitted to add a global record by means of a suspense transaction, then the possibility would exist of simultaneous and independent additions of the same record at different sites. On the other hand, if the addition of global data were performed through broadcast transactions, then a critical aspect of daily business, the addition of a new part, would depend on the availability of all sites.

The solution reached was the creation of a range file that restricts the add capability of any site to a nonoverlapping set of values. The range file is replicated at all sites, and adds can be performed using the suspense mechanism so long as the key value falls within a valid range. Following our example, each site is permitted to add parts within a unique range of part numbers.

The range file is maintained using broadcast transactions.

```
Initiating                             Remote
   Site                                 Site(s)


+-----+                               +------+
|     |                               | A  S |
|  S  |                               | S  E |            ////////////
|  U  |     (2)      ---------------->| S  R |  (3)       | EMPACT   |
|  S  |-------------\                 | O  V |-------->   | DATABASE |
|  M  |                               | C  E |            |///////////
|  O  |              -----------------| I  R |
|  N  |<------------\      (4)        | A    |
|     |                               | T    |
+-----+                               | E    |
   ^  |                               |      |
(1) |  | (5)                          +------+
    |  v
+-------------------------------------------------------------+
| Suspense |      Suspense record key      |      msg        |
|  file    |Counter value  Sequence Number|                  |
|          |     n               0         | New Trans       |
|          |     n+1             0         | New Trans       |
|          |     n+2             0         | New Trans       |
+-------------------------------------------------------------+
```

Figure 10—Suspense processing: (1) SUSMON obtains transaction *n*0 from the suspense field; (2) SUSMON establishes communication with server at remote site and sends transaction message; (3) server updates local copy of database; (4) server replies to SUSMON; (5) SUSMON updates the bit mask of *n*0 in the suspense file and begins processing transaction (*n* + 1)0.

```
Initiating                             Remote
   Site                                 Site(s)


+-----+                               +------+
|     |                               | A  S |
|  S  |                               | S  E |            ////////////
|  U  |              ---------------->| S  R |            | EMPACT   |
|  S  |-------------\                 | O  V |-------->   | DATABASE |
|  M  |     (2)                       | C  E |            |///////////
|  O  |              -----------------| I  R |
|  N  |<------------\                 | A    |
|     |                               | T    |
+-----+                               | E    |
   ^  |                               |      |
(1) |  | (3)                          +------+
    |  v
+-------------------------------------------------------------+
| Suspense |      Suspense record key      |      msg        |
|  file    |Counter value  Sequence number|                  |
|          |     n               1         | New Trans       |
|          |     n               2         | 2nd request     |
|          |     n               3         | 3rd request     |
|          |     n+1             0         | New Trans       |
+-------------------------------------------------------------+
```

Figure 11—Suspense processing: (1) SUSMON polls the suspense file and observes that the sequence number in the suspense key of the first record identifies the first record of a multirequest transaction; SUSMON recognizes that the unit of work demanded in this case requires that all entries for transaction *n* be processed before completion is reached; (2) SUSMON processes the requests serially in the usual manner; (3) if the server has replied successfully to all requests involving transaction *n*, completion has been reached, the bit mask of *n*1 is updated and SUSMON begins processing transaction (*n* + 1)0.

Paired with suspense processing, the range file concept satisfies all requirements for uniqueness, site autonomy, and user friendliness.

The interplant transfer of material posed a somewhat different problem. As noted earlier, the interplant transfer data are considered semiglobal because the information is relevant only to the sites involved in the exchange of material, which may be two or more sites but need not be all. The events of a typical interplant transfer of material are as follows: one site requests material from another in the form of a requisition; the supplying site acknowledges the requisition and determines whether there is sufficient available inventory to fulfill the request; if so, the supplying site ships the requested material, and upon receipt the requisition is closed.

A decision was made to implement the interplant transfer functions, or MART subsystem, by replicating the interplant transfer data at the sites involved in the transaction and using the suspense mechanism to move the data from one site to the other.

Suspense transactions in the MART application are composed of potentially multiple requests. The proper processing of these requests at the destination site requires a precise sequencing at the sending site. The internal serialization of a multiple-request transaction can be obtained by incorporating a sequence number into the key of the suspense record; the key is now formed by a counter that contains the value of the SUSMON-TRANS-ID and a sequence number that directs internal order. SUSMON then treats all records associated with a particular counter value as a single transaction but sends the requests, one at a time, to a specified server. The result of these modifications was the evolution of SUSMON from a tool intended solely for broadcasting transactions to a general-purpose transport mechanism.

Figures 10 and 11 describe the differences between a standard SUSMON transaction and a MART SUSMON transaction in terms of the SUSMON-TRANS-ID.

Error determination and handling in a distributed environment requires a level of clarity and coordination that may be unnecessary in a nondistributed setting. A distributed database without a strategy for identifying and resolving errors is singularly vulnerable. SUSMON was chosen as central coordinator of error detection within the distributed EMPACT.

Error Processing

SUSMON's interpretation of error conditions determines the subsequent suspense processing to be followed. SUSMON identifies an error as belonging to one of two distinct categories; either the error is retryable or nonretryable. The selection of a category is based on the following criteria:

1. Retryable.—The error was caused by an unanticipated system restriction or problem. Possibilities include communications disruption between SUSMON and a remote server, site unavailability, or isolated system failures within the remote system.
   Solution.—The sending SUSMON will report the problem, defer any suspense processing to that site for an ordained amount of time, continue processing to other sites, and eventually retry the transactions.
2. Nonretryable.—The error was related to data inconsistency and was reported by the remote server that had attempted to satisfy the request. Possibilities include failure during the remote server's old-data/new-data check or any other consistency evaluation, or media damage such as an uncorrectable parity error within a record accessed by the remote server.
   Solution.—The remote server will report the problem on the remote site, and the sending SUSMON will do likewise locally. SUSMON will indefinitely suspend all activity directed to the site reporting the failure. Resolution of the problem will require a degree of human intervention.

Any problem reporting is done to a hard-copy logging console. It is, moreover, essential that SUSMON suspend all activity to any site that is in error; the serial interdependence of transactions requires the serial processing of any suspense file requests destined to a site. A nonretryable failure of a single transaction to a site effectively interrupts the serial flow of data. Processing to all unaffected sites can continue, since the situation does not imply an error condition outside that reported in the problem site.

EXPERIENCE

The development schedule for distributed EMPACT spans a period of 2 years. The project was commissioned in January 1981 and it was not until September of that same year that the preliminary design was agreed upon. External and internal specifications were completed 3 months later, and in January 1982 coding began.

The development of SUSMON and the conversion of MART to use SUSMON were completed in April 1982. The conversion of the item master subsystem and bill of materials subsystem, along with the restructuring of the database are scheduled to be finished in December 1982. MART and SUSMON were installed in May 1982 for limited production use. Until the final conversion effort is completed, weekly tapes are being sent to all the sites to maintain concurrent copies of the global information.

The results of the limited production use of MART and SUSMON are encouraging. The software has responded well to the demands placed on it, and the assumptions made during the design phase have proven accurate. Operationally, however, the need for a centrally located network management function has become apparent; events that require the cooperation of all the different sites, such as the addition of a new site to the network, are very difficult to manage manually in a distributed environment.

This need was evidenced during the first attempt to quiesce a three-site EMPACT network. Quiescing a network requires that all servers across the EMPACT network be denied access to their suspense files, and that each SUSMON empty its suspense file of any outbound transactions.

The strategy chosen for this first attempt was to coordinate the activity over the telephone. The exercise demonstrated that the operational complexity of a distributed application demands a high level of training and skill at both the system manager and computer operator positions.

To relieve these difficulties, software was developed to aid in the management of the network, especially in the areas of control and configuration. Its capabilities include

1. The addition and deletion of sites to and from the network
2. The quiescing of all global transactions
3. Inquiry into the status of global-transaction processing throughout the network

It should be noted that SUSMON was used to implement the network management software.

CONCLUSIONS

Distributed EMPACT is an example of an application where the requirements of a business call for a distributed solution. The application is considered successful because the structure and organization of the database and software closely parallel the structure and organization of TANDEM's business environment.

The design of distributed EMPACT illustrates some of the techniques that can be used in a distributed database application, and the actual implementation of distributed EMPACT demonstrates the feasibility of developing a truly distributed application on the TANDEM T/16 system.

ACKNOWLEDGMENTS

We would like to acknowledge the contributions of Vince Sian and Jim Gray to the initial design of Distributed EMPACT. Further thanks are due to Betty Buhr, Rick Burgess, Bob Plummer, Kathy Welch, and to the entire Manufacturing Information Systems staff for their ideas and advice. We also wish to recognize Carl Goldman and Mike Sanders for the management support that they provided. We are grateful for the help and guidance given us by many members of the TANDEM Software Development and Software Education groups, particularly Andrea Borr, Bob Jones, and Jim Collins. Finally, we would like to thank Jim Gray and Rikki Westerschulte for their editorial advice.

REFERENCES

1. Borr, A. "Transaction Monitoring in ENCOMPASS: Reliable Distributed Transaction Processing," TANDEM Technical Report TR 81.2, 1981.
2. Katzman, J., and R. Taylor. "GUARDIAN/EXPAND, a Nonstop Network," TANDEM Technical Report, 1978.
3. Selinger, P. "Replicated Data." In I.W. Draffan and F. Poole (Ed), *Distributed Data Bases, An Advanced Course.* Cambridge: Cambridge University Press, 1980, pp. 223–231.
4. Smith, L. "Designing a Network-Based Transaction Processing System." SEDS-002, TANDEM Computers Incorporated, 1982.

Dynamic replication, an overview

by T.P. DANIELL, R.C. HARDING JR., and S.H. NAUCKHOFF
IBM
Palo Alto, California

ABSTRACT

Dynamic replication is a new technique for managing data in a distributed processing environment. It is a simple yet powerful scheme that addresses the major problems of distributing databases.

This paper discusses dynamic replication and database distribution in three parts: First, several aspects of database distribution common to various techniques are described. Second, some capabilities of dynamic replication are emphasized, inasmuch as they are not available with other techniques. Third, an overview description of dynamic replication is given.

INTRODUCTION

A key characteristic of distributed data environments today is diversity—diversity of applications requirements and diversity of distribution techniques. Some of today's distributed data techniques require the user to bury data management function in application programs, operational procedures, or application restrictions.[3] Other techniques require synchronized clocks, centralized locking, predefined transaction work sets, or that all or a majority of nodes vote on each node's updates.[1,2]

Dynamic replication is a new technique for managing data in a distributed-processing environment. It is a simple yet powerful scheme that addresses the major problems of distributing databases. It can satisfy a wide variety of application requirements and generally it subsumes or improves upon the consistency, performance, and availability characteristics of other techniques.

This overview consists of three parts. First, several aspects of database distribution common to various techniques are discussed. Second, some capabilities of dynamic replication are emphasized inasmuch as they are not available with other techniques. Third, an overview description of dynamic replication is given.

ASPECTS OF DATABASE DISTRIBUTION

This section covers

1. Data access needs from the point of view of an enterprise
2. Teleprocessing network design problems
3. Various ways data access needs can be satisfied via the teleprocessing network using current data management technology

Data Access Characteristics

Data are information abstracted from some aspect of business important to an enterprise. The challenge is to give end users access to the data with an availability, performance, and cost commensurate with their business requirements. System considerations that unnecessarily interfere in the use of the data can be costly to the enterprise.

The end users of the data are assumed to be geographically dispersed. They need access to the data. Neither end users nor application programmers should be required to be aware that the data are distributed or to know their location.

The need for database management is accepted. Access to the data must be controlled to ensure authorization of access and to ensure the consistency and integrity of the data.

Additionally, data accesses in a distributed environment may have certain affinities, as follows:

1. Geographic affinity.—Accesses to a given data item tend to cluster geographically. Data items that have been accessed at a given node are more likely to be accessed again at that node than other items are. This is the basis for improved performance and availability for all distributed data schemes.
2. Temporal affinity.—Accesses to a given data item tend to cluster in time. Data items that have been accessed recently are more likely to be accessed in the near future than data items not recently accessed.

The dynamic replication technique is designed to take advantage of these affinities.

The node at which accesses for a given data item tend to cluster is called the *affinity node.* With dynamic replication the affinity node for a given data item need not be known ahead of time; furthermore, the affinity node for a given data item may vary with time.

Teleprocessing Network Design

For the purposes of this discussion, the teleprocessing network design includes determination of the number of nodes, their geographic placement, the communication paths and capacities between nodes, the connection of user terminals to particular nodes, determination of which applications will execute at which nodes, and so on. The dynamic replication scheme is not concerned with this design problem; dynamic replication will work with any teleprocessing network. The discussions that follow assume that the teleprocessing network has been designed already.

Aids are available today to help users design their teleprocessing networks. The communication and processing load that results when dynamic replication is used to manage the distributed data is likely to be quite different from the load when other techniques are used. Therefore, the network design aids used should allow for the capabilities of dynamic replication in their design algorithms.

The dynamic replication technique is independent of the teleprocessing network topology. Dynamic replication uses current communications technology to transfer data and control information between nodes.

Data Distribution Technology

Distributed data technology has been categorized according to various attributes, including

1. Data location (centralized, partitioned, and replicated)
2. Degree of data sharing (centralized, decentralized, and distributed)
3. The degree to which database management control is provided network-wide (distributed data and distributed database)
4. Type of data access (transaction shipping, function shipping, data shipping)

The dynamic replication technique provides distributed database management based on the shipping of replicated data.

CAPABILITIES UNIQUE TO DYNAMIC REPLICATION

Dynamic replication generally subsumes the essential characteristics of other data distribution methods. Furthermore, dynamic replication has significant capabilities that are not available with the other methods. These capabilities include

1. Increased flexibility in the use of node resources
2. Trade-offs among data access performance, data consistency, and availability of data

Flexible Use of Node Resources

With dynamic replication, adding disk capacity, CPU power, or additional CPUs can be accomplished with less disruption at other sites than other methods cause. This flexibility allows the user to accommodate application growth and changes in load patterns easily. Growth beyond the top of the CPU line (central or remote) can be accommodated.

Disk capacity at a dynamic replication node affects performance and availability rather than the functional capability available at the node. With other methods, enough storage must be available at a node to hold the replica or partition for use at that node. If there is not enough storage available, then the application cannot run. If additional storage is available, it cannot be effectively used to improve the application performance. In contrast, dynamic replication can use whatever disk storage is available for replica storage. The more storage available, the higher the probability that a given data item will be available locally when it is next required. Adding disk storage at a dynamic replication node improves performance and data availability at that node.

It is relatively easy to add another CPU to a network of dynamic replication nodes. With dynamic replication, only the nodes that will directly relate to the new node need be aware of the new CPU. No special update procedures need to be devised as they must in the case of static replication. The affinity information at each node (in the application programs, declarations, or system programmer exits) need not be revised as it must for partitioned-data schemes.

Adding another node is simply a performance and availability factor with dynamic replication. Having two small CPUs next to each other rather than one larger CPU is a valid possibility. CPUs should be located (and terminals attached to them) to take advantage of the geographic and temporal affinity of the database accesses. A poor topographical choice is one for which different nodes tend to access the same data at the same time and with high currency requirements. If this consideration is ignored, adding a new node might reduce rather than increase performance.

Consistency/Performance/Availability Tradeoffs

Different applications have different consistency requirements for the data they access. In dynamic replication, data consistency is defined in terms of the data currency and data stability requirements of an application. This is illustrated by the EMPLOYEE database shown in Figure 1. An application

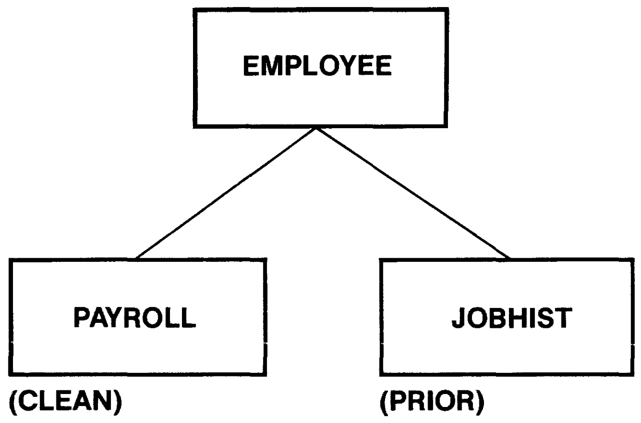


Figure 1—Data consistency

program that is updating the payroll information (to compute a raise, for example) for a given employee needs to work with the most current value of the PAYROLL data (called *clean* in dynamic replication). However, another application program that is preparing a report on job statistics by location is probably content with JOBHIST data values that may not be the most current (called *prior* in dynamic replication).

The dynamic replication scheme allows the user to make a tradeoff among performance, availability, and data consistency. This tradeoff can be made for each application. Thus, database integrity is ensured by providing clean data to those applications that require it; on the other hand, performance and availability gains can be achieved for those applications with less demanding currency requirements.

The extent to which a user can make tradeoffs among performance, availability, and consistency is unique to dynamic replication among current data distribution schemes. With static-replication or partitioning schemes, this tradeoff is made once and for all when the scheme is chosen—and all applications must live with the tradeoff.

Dynamic replication also has options that allow the user to make a tradeoff between performance and availability for each location. Furthermore, a tradeoff can be made, for each data item, between performance and availability at an item's affinity node versus other nodes.

Unlike partitioning, dynamic replication does not require the user to provide any affinity information. The dynamic replication data placement algorithms are adaptive; data tend to move to the locations where they are accessed. However, if data affinity information is available, it can be used by the dynamic replication technique. Dynamic replication will use the affinity information to improve performance and availability at the affinity node (for a given data item). The other benefits of dynamic replication are not sacrificed.

THE DYNAMIC REPLICATION TECHNIQUE

In this section the components of a dynamic replication system are defined and their interactions are described.

Dynamic Replication Components

The major components of a dynamic replication system are shown in Figure 2. The components that are new in a dynamic replication system are defined in this section.

Every node in the network has the capability of storing some number of data items. Copies of data items are dynamically created at a node as required to support the processing that occurs at it.

There is a replica file manager at each node, which is responsible for the data items stored there. Also at each node is a replica distribution manager (RDM). The RDMs at the various nodes are responsible for controlling the distribution of data. RDM intercepts data access calls and makes requests to other nodes, if necessary, for the data and authorities required to honor the data access call.

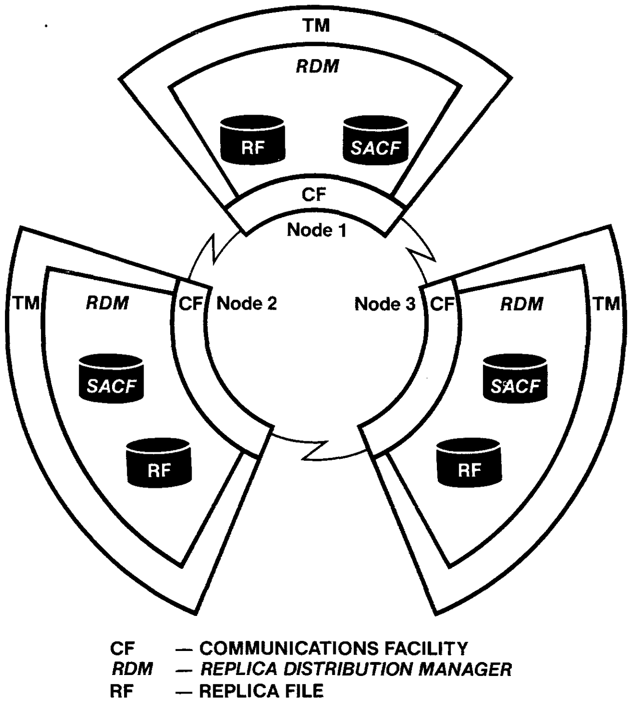


Figure 2—Components in a dynamic replication network

Each dynamic replication node can store some number of data items. These data items may be held uniquely at the node or may be replicas of data items held at one or more other nodes. These replica data are stored in the *replica file.* Data that are required to honor an application database call can be requested from another node and stored in the replica file. Existing data in the replica file may have to be removed to make room for the new data.

The RDM insures data integrity and consistency by operations at two levels. Access conflicts between application programs running in the same node are managed by using locks held on behalf of the application programs. Data item locking is managed independently at each dynamic replication node.

Access conflicts between application programs running at different nodes are managed using control information in a format called *dipoles.* A dipole describes the relationship between two nodes with respect to one or more data items. Each of the two nodes holds one half of the dipole. That half describes the other node's status for the data items. It describes, for example, whether the other node has a copy of the data items, the currency (clean or prior) it is assuming for its copy, and whether it is possibly making updates to the data items.

The set of all dipole halves held at a node for a data item provides the RDM with the necessary information about the status of the data item. This information is called the *data state* for the item. Data state information is held in the status and control file (SAC file). The RDM can determine from the SAC file whether or not applications at its node can safely reference or update a data item, as well as find in it information about the currency of the data item.

Any time an application program inserts, deletes, or replaces a data item, a record of the update is placed in the SAC file. Update status information is stored in the SAC file until it is required for transmission to other locations. It is not

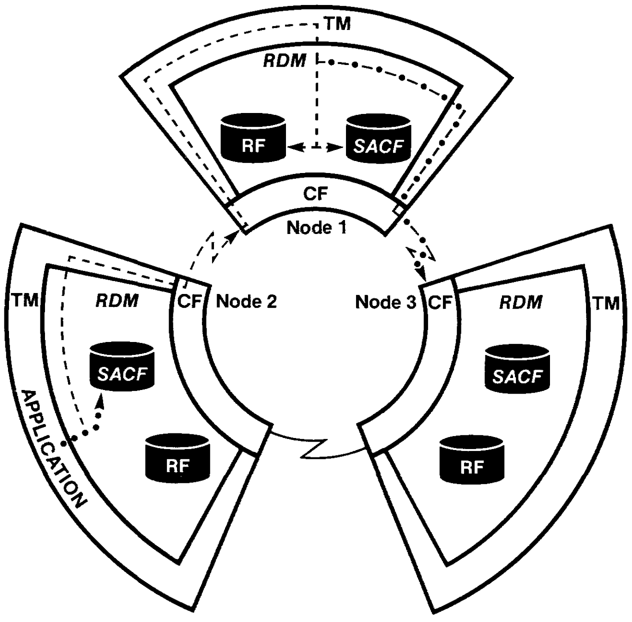


Figure 3—Application request processing

necessary to inform other locations of the updates as they occur.

Application Request Processing

The dynamic replication processing for an application program database request is diagrammed in Figure 3. When a database request is made at node 2, the request is intercepted by the RDM. The RDM interrogates the SAC file to determine whether the node holds the necessary data and authorizations to satisfy the request. If it does, then the request is simply passed on to the replica file manager and the results are returned to the application program.

On the other hand, the RDM may determine that one or more existing dipoles conflict with the requirements of the application request. For example, a related node may be authorized to update the same data item that this application wants to update. In this case the RDM makes a request for a dipole change to the node (or nodes) with which it shares a conflicting dipole.

The request is passed through the communication facility to the RDM at the related node. The related RDM must interrogate its SAC file to be sure no dipoles exist with still other nodes that conflict with the request. If necessary, requests are sent to other RDMs to change their conflicting dipole halves.

When this conflict processing is complete, the related RDM can change its dipole half and give a positive response (with data, if required) to the originating RDM. The originating RDM stores any received data in its replica file, records its half of the new dipole in its SAC file, and passes the application request on to the replica file manager.

Once data are stored in the replica file, they normally stay there until the space is needed for some other data item. Thus the data are available for satisfying subsequent data requests without further network interactions. This contrasts with partitioned database schemes, where data must be re-retrieved for subsequent requests.

Several examples of dynamic replication processing will now be described, using the database illustrated in Figure 4. The replica file at node 1 contains four records (keys A, J, K, and S). Node 2 has replicas of records A and J in its replica file. Node 3 has replicas of records J and K. Record J has been modified at node 3.

Example 1: Shared clean data. The ability to share clean data among nodes is illustrated with record A. Node 1's SAC file reveals that node 1 understands that both node 2 and node 3 have copies of record A, which each is assuming to be clean. This is shown as a status of SHARE in the figure. If clean data are shared, then none of the parties involved can modify the data without first getting the other nodes to change their view (as represented in the dipole half) of the data state.

A dipole describes the relationship between two nodes for a data item. Each of those nodes may have additional relationships for the same item with other nodes as long as the additional relationships do not conflict. For example, node 2 is aware that it is sharing clean data with node 1 but is unaware that node 1 has also shared the record with node 3.

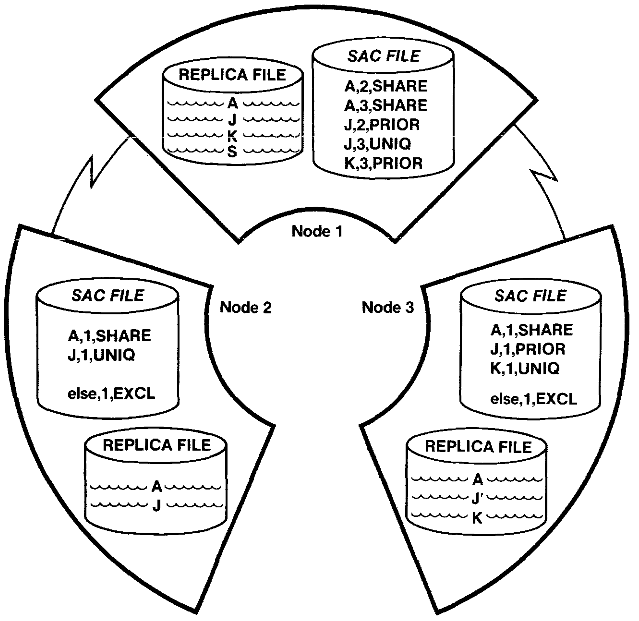


Figure 4—A dynamic replication example

If an application program attempts to read record A at node 2, then the access request will be passed to the replica file manager since node 2 is sharing clean data for record A with node 1. No network interactions are required.

Example 2: Updates and prior data. A dipole situation in which data updates can be taking place is shown for Record J. The dipole half for Record J at Node 2 shows that Node 1 has a *unique clean* view of Record J. This tells Node 2 that Node 1, or some node related to Node 1, may be making updates to Record J. Thus, Node 2 must assume that its replica of Record

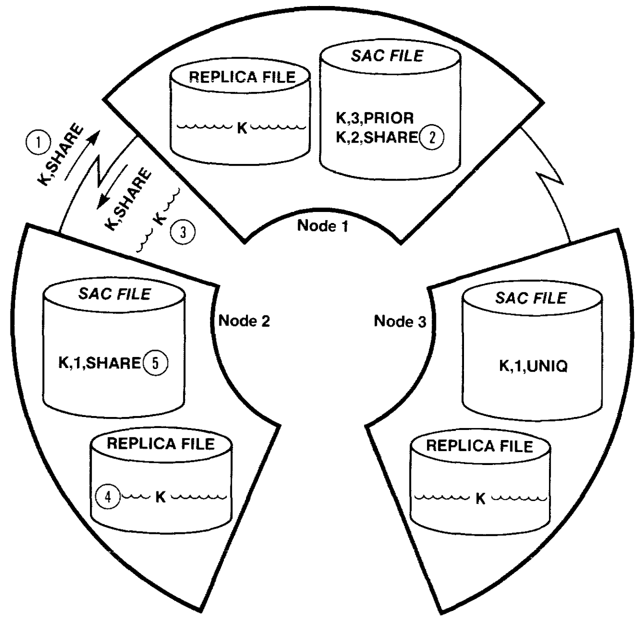


Figure 5—Obtaining clean data

J is prior data. Update activity cannot be permitted at Node 2 unless a change to the dipole with Node 1 is negotiated.

Node 1 has passed along the unique clean view of record J to Node 3. Node 3 has, in fact, already modified record J.

If an application program attempts to read Record J at Node 2, the access request will be passed to the replica file manager only if the application has indicated that it is accepting prior data. The data at Node 2 must be considered prior, since Node 1 (or a node attached to Node 1) may be making updates to Record J.

Example 3: Obtaining clean data. Figure 5 illustrates a retrieval call for Record K made by an application program at Node 2. For this example, the program is assumed to be requiring clean currency for the data it accesses.

Node 2 does not have a specific dipole half for Record K with any node. However, the last dipole half in the SAC file at Node 2 indicates that all records that are not otherwise described by dipole halves are assumed to be held exclusively at Node 1. Data state management is therefore required before the application call can proceed.

The processing is outlined as follows:

1. A request is made by the RDM at Node 2 to the RDM at Node 1 for the data to be shared with clean currency.
2. The only dipole half at Node 1 for Record K is with Node 3. The dipole half with Node 3 does not conflict with the Node 2 request. Node 3 is assuming that its copy is prior; therefore it cannot be making updates to Record K. The new dipole half with Node 2 is recorded in the SAC file at Node 1.
3. Node 1 sends a positive response to the request, along with a copy of Record K. This response acknowledges that Node 1 (and possibly other nodes related to Node 1) is still assuming its copy is *shared clean.*
4. The RDM at Node 2 stores its copy of Record K in its replica file.
5. The RDM records the new dipole half in the SAC file.

Now the original application program request can be passed on to the replica file manager.

Example 4: Retrieval with intent to update—No conflicts. Figure 6 illustrates a retrieval call for Record K with intent to modify the record. The call is made by an application program at Node 2. Since Record K is not replicated at Node 2, dynamic-replication data state management is required.

The processing is outlined as follows:

1. A request is made by the RDM at node 2 to the RDM at Node 1. Since the application has specified an intent to modify the record, a unique clean copy of the data is requested.
2. The only dipole half for Record K does not conflict with the request; therefore the RDM records the new dipole half with Node 2 in the SAC file.
3. The RDM sends a positive response to the request, along with a copy of Record K. This response acknowledges that Node 1 (and all other nodes related to Node 1) is now assuming its copy of Record K to be prior.
4. The RDM at node 2 stores its copy of Record K in its replica file.
5. The RDM records the new dipole half in the SAC file.

Now the original application program request can be passed on to the replica file manager.

Example 5: Retrieval with intent to update—Existing conflicts. Figure 7 illustrates a retrieval call for Record J with

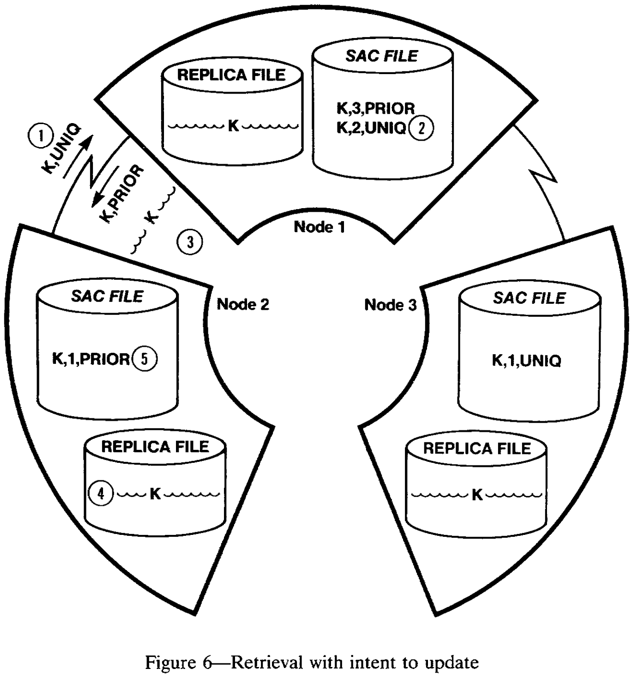


Figure 6—Retrieval with intent to update

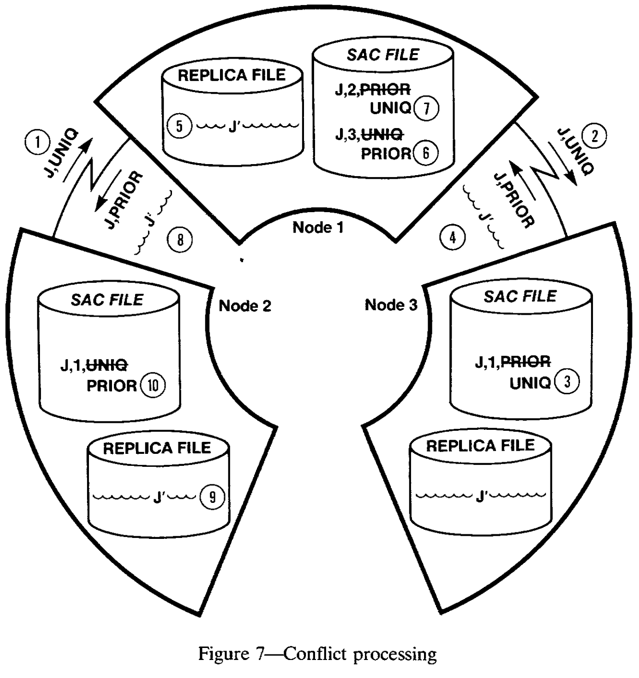


Figure 7—Conflict processing

intent to modify the record. The call is made by an application program at Node 2. A copy of Record J is already held at Node 2, but the dipole half indicates that Node 1 (or some node related to Node 1) may be making changes to Record J. Dynamic-replication data state management is required.

The processing is outlined as follows:

1. A request is made by the RDM at Node 2 to the RDM at Node 1. Since the application has specified an intent to modify the record, a unique clean copy of the data is requested.
2. Node 3 has the unique clean view of Record J. This conflicts with the request from Node 2.
   The RDM at Node 1 requests Node 3 to surrender the unique clean view of Record J. The RDM at Node 3 can retain its replica at an assumed currency of prior.
3. The Node 3 RDM surrenders its view as required. This action may have to wait for the completion of an application program running at Node 3 if the application program is positioned on Record J, has locked it, or is modifying it.
4. The Node 3 RDM eventually gives a positive response to the request from Node 1. This response acknowledges that Node 3 is now assuming its copy of Record J to be prior. Since at least one change to Record J has been performed at Node 3, the most recent value of the record is sent with the reply.
5. The Node 1 RDM saves the new value of Record J in its replica file.
6. The Node 1 RDM records the Node 3 dipole half change in its SAC file.
7. The Node 1 RDM records the new dipole half for Node 2 in its SAC file.
8. The Node 1 RDM sends a positive response to the original Node 2 request, along with a new copy of Record J. This response acknowledges that Node 1 (and all other nodes related to Node 1) is now assuming its copy of Record J to be prior.
9. The Node 2 RDM replaces its copy of Record J in its replica file.
10. The Node 2 RDM records the Node 1 dipole half change in its SAC file.

Now the original application program request can be passed on to the replica file manager.

Conformation

When a database update is made at one node, this update may need to be reflected at other nodes in the network. This process is called *conformation,* since the networkwide database is made to conform to an update made at one location. The conformation process is diagrammed in Figure 8. Since dynamic replication insures that conflicting data state dipoles never exist in the network, database integrity is not affected by when, or even whether, conformation occurs. Performance and availability, however, are affected by the scheduling of conformation.

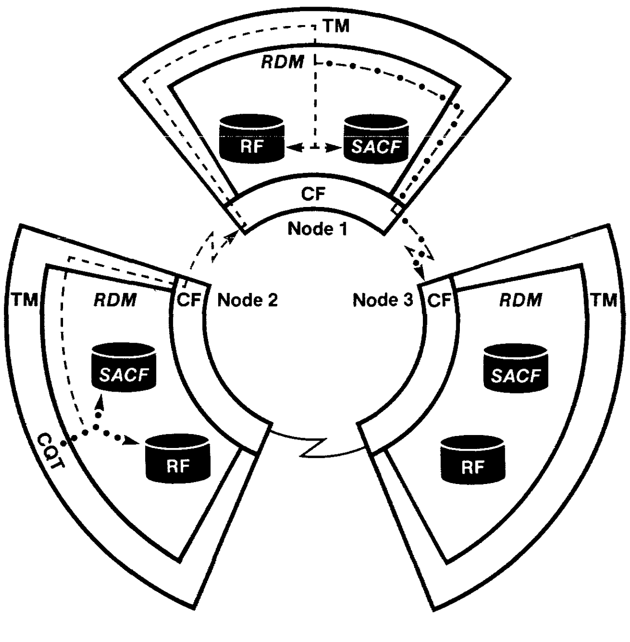


Figure 8—Update conformation

Conformation can be initiated in several ways:

1. When the authority to update a data item is given up by a node, any modifications to the data item made by the node are conformed at that time. This may occur in response to a request from another node or because space in the replica file is needed for some other data item.
2. Conformation can result from the initiation of the dynamic-replication *change queue transaction.* The change queue transaction can be initiated by any mechanism available for scheduling application transactions at the replica node: time-of-day, time interval, operator command, work to be done, and so on.

The first step of conformation is to read the update status information in the SAC file and transmit the modified data to one or more nodes that are related to this node for the data item. The replica and SAC files at the related nodes are made to reflect the change. Each of the related nodes can in turn conform the updates to still other nodes with which it is related.

Data Distribution Network Topology

Previously it was stated that dynamic replication is independent of the teleprocessing network topology. The *teleprocessing* network topology describes how the nodes are related for the purposes of communication. However, another network topology is central to the dynamic replication scheme—the *data distribution* network topology. The data distribution topology describes how the nodes are related for the purpose of distributing a particular database.

If a node is not otherwise related to other nodes for a

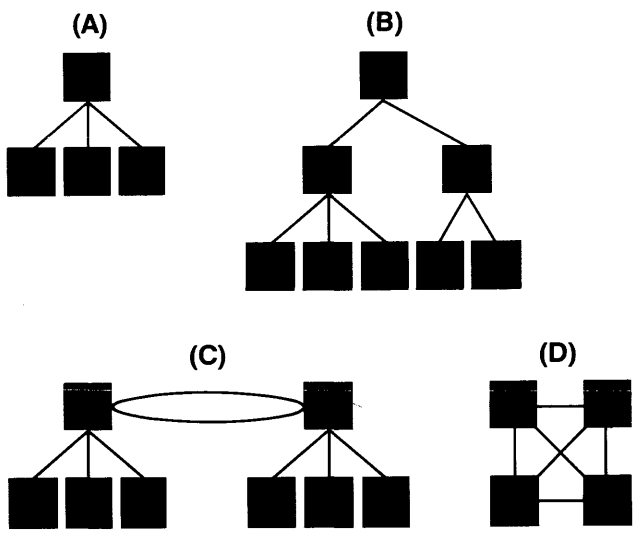


Figure 9—Data distribution network topology

particular data item, then the data distribution network determines to which node it will direct its request for that data item. Some typical data distribution network topologies are shown in Figure 9. The network in Figure 9(a) is a two-level hierarchy. This means that all nodes that require any data item will request it from one node (called the *primary* node). The primary node will intermediate with the other nodes as required. A full-peer network is diagrammed in Figure 9(d). In this case each node directs its requests to a node that is determined by some criterion such as key range. That node then intermediates as required with other nodes. The other networks shown lie between these two extremes.

Different choices of data distribution topology can be made for different databases that are distributed on the same communications network. The choice affects performance and availability of the data as well as the degree to which centralization of the database administration function is to be imposed.

CONCLUSION

Dynamic replication manages to avoid the disadvantages of other methods for distributing data while maintaining their advantages:

1. It improves availability and performance for data that are held at the node where it is accessed.
2. It increases availability and performance for application programs that can accept data which are possibly not the most current.
3. It provides data location transparency; application programmers and end users need not be aware of data location or even of a data-partitioning algorithm. The database appears to be the same as in the single-system approach.
4. A partitioning algorithm is not required in dynamic replication. Dynamic replication automatically moves data to the location(s) where they are being accessed.
5. It provides good performance for programs that access the database uniformly. For example, a given node can have a copy of every data item in the database. These data need not be the most current. A summary program can run at such a node with minimal performance degradation.
6. No special update procedures or windows are required; updates are managed networkwide by dynamic replication.
7. Database integrity is ensured by dynamic replication in cooperation with the replica file managers. Multiple updates are prevented. Application programs receive data as current as they require.
8. The teleprocessing network may be unstable. Unlike some methodologies, dynamic replication does not require that communication with all nodes, with a given percentage of nodes, or to any particular node be available at transaction termination time.

The advantages of dynamic replication result from the maintenance of control information on a data item basis in the form of dipoles in the SAC file. The chief disadvantages of dynamic replication are the costs of storing and accessing that information:

1. Disk storage is required at all locations for the SAC file. Furthermore, disk storage for data is greater for dynamic replication than for partitioning. However, disk storage for data for dynamic replication is less than it is for static replication, or about the same.
2. The access time for data that are stored locally may be longer for dynamic replication than for some other methods. This effect may be more than offset by the reduced need for accessing the data via the network compared to partitioned data schemes.

The dynamic replication technique has wide application. It is a simple but powerful technique that generally subsumes and improves upon the consistency, performance, and availability characteristics of other schemes. It also provides significant operational, availability, and performance benefits not otherwise available.

Dynamic replication solves many of the basic problems of distributing data by managing the database networkwide. This unified approach can be used to improve application programmer productivity, reduce operational costs, and improve the availability of data and the performance of database accesses.

REFERENCES

1. Bernstein, P.A., and N. Goodman. "Concurrency Control in Replica Distributionbase Systems." *Computing Surveys,* 13, no. 2 (June 1981), pp. 185–221.
2. Kohler, W.H. "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems." *Computing Surveys,* 13, no. 2 (June 1981), pp. 149–183.
3. Martin, J. *Design and Strategy for Replica Distribution Processing.* Englewood Cliffs, N.J.: Prentice-Hall, 1981.

Local query translation and optimization in a distributed system

by EMMANUEL ONUEGBE and SAID RAHIMI
Honeywell Corporate Computer Sciences Center
Bloomington, Minnesota
and
ALAN R. HEVNER
College of Business and Management
University of Maryland
College Park, Maryland

ABSTRACT

A new query translation and optimization algorithm is presented. The algorithm is being implemented as the local query translation and optimization technique of Honeywell's Distributed Database Testbed System (DDTS). The algorithm translates local queries expressed in representational schemas (relational) to their equivalent internal schemas (network). The technique is new in that it does not translate each relational command in isolation, but rather attempts to find a collection of relational commands for which an optimized sequence of CODASYL DML commands can be generated. The optimization minimizes the number of disk accesses by taking advantage of the access paths available to the CODASYL local database management systems and the relationship information of the variables used in the relational commands.

INTRODUCTION

In a distributed database system, portions of the data are stored at different nodes in the network. All of the data are regarded as one database, because internode communication and resource sharing permit access to the data resident at other nodes (subject to the access constraints of the network). A system-wide discipline is needed to enforce or facilitate security, data access, resource use, operating procedures, database definition, and data and program transfer among system nodes. At each node, a local database management system provides access to the resident data. These local database management systems may be heterogeneous within the network. It is necessary, therefore, to provide users with a general transaction processing interface that processes user requests while hiding the heterogeneity of the system from the user.

The ANSI/SPARC Study Group on Database Management Systems[1] has proposed a framework consisting of three levels of schema definitions. These levels consist of:

1. external schema: the description of the user view
2. conceptual schema: the description of the logical view of the data
3. internal schema: the local DBMS implementation of the database.

In order to model distributed databases, a recent proposal[2] generalizes the ANSI/SPARC framework to five levels (see Figure 1). This proposal extends the internal schema to:

3.1. global representation schema: the description of the global representation of data
3.2. local representation schema: the description of the database at a given node
3.3. local internal schema: the local DBMS implementation of the database.

A different data model may be needed to perform the functions at each different schema level. This means that multischema architectures may entail multimodel architectures. At the external schema level, for instance, different data models may be used to describe different user views of the data. The conceptual schema level requires a semantic data model that is not cluttered by implementation details. The representation schema levels require a data model that possesses powerful data manipulation operators (e.g., the relational data model). The internal schemas are defined in terms of the models supported by the Local DBMSs (e.g., the CODASYL DBTG Model). Thus, in a DDBS that incorporates a multischema/multimodel architecture, various phases of translation are needed in order to express a user view of data in one or more local DBMS representations of that data. Also, in order to achieve reasonable system performance, optimization techniques must be employed in these translations and for access to the data at the local nodes.

The purpose of this paper is to describe a general method of translating and optimizing a database query from the local representation schema level to the local internal schema level. A number of other papers have described methods of query translation and optimization from the external level to the global representational level[3,4] and from the global representational level to the local representational level.[5] The methods described in these papers and in this one are being implemented in the Distributed Database Testbed System (DDTS) at Honeywell's Corporate Computer Sciences Center.[6,7] The external level and conceptual level of the database are described by the Entity-Category-Relationship Model.[4] The representational schemas in DDTS use an extended relational model of data.[8] The local database management systems are Honeywell IDS/II systems, which use a CODASYL network model.[14] Thus, the goal of this work is to describe methods of translating and optimizing relational subqueries at local nodes with CODASYL database management systems.

Other work on this problem is taking place at several research centers where distributed database systems are being constructed. The differences in translation and optimization methods among these centers are due to the different data models used at the representational levels and the local internal level of the systems. A recent paper by Dayal and Goodman[9] addresses a translation and optimization environment similar to that of DDTS. A major difference exists in that their proposed methodology interpretively generates a database access strategy for each query entered into the system. A cost formula is optimized in order to derive efficient local processing strategies. Although this method produces an optimal access strategy, the complexity of this derivation is exponential with respect to the query size and thus could be quite costly to execute at run time. In contrast, the object of the methodology in this paper is to rapidly recognize only the query access patterns that are potentially beneficial for optimization. These patterns are preprocessed and are readily available for execution.

The second section discusses the translation of subqueries from the relational data model to the CODASYL network data model. During this translation, the subquery is optimized to take the best advantage of the implemented access paths on the local DBMSs. In the third section we discuss methods for recognizing these optimization opportunities. The fourth section describes the implementation of the translation and opti-

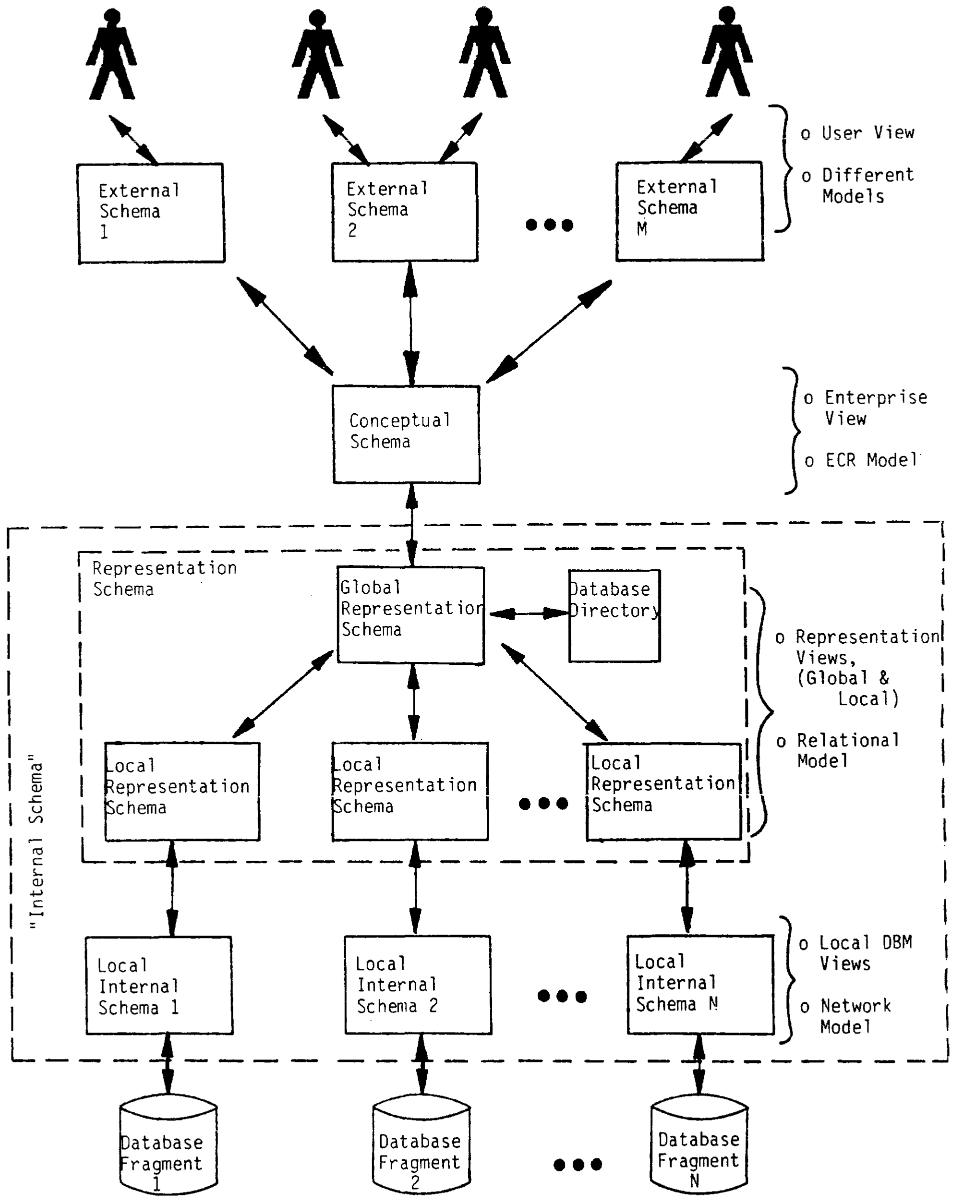


Figure 1—Five-schema architecture

mization methods in DDTS. We conclude the paper by briefly describing planned performance studies of our methodology and suggesting future extensions.

LOCAL TRANSLATION AND OPTIMIZATION

A query, received by a local node for processing (compilation and/or execution), is a list of relational commands on a set of base or temporary relations in a database. Local queries are composed of relational operations that can be executed completely on a local relational schema. Since we assume that the local database management systems are based on CODASYL specifications (IDS/II),[10] the system must transform local relational schemas into local network schemas and relational commands into an equivalent sequence of network DML statements, which can be executed on the network schemas. The transformation of schemas (relational to network and network to relational) is done at database design time and is stored in local data dictionaries for use in later translation and optimization.

This transformation is based on a one-to-one correspondence between relations and record types and between attributes and data items.[11] Stated more formally, the relational to network transformation is:

Let S be a relational schema with K relations, then:

1. For each relation Ri, $1 \leq i \leq K$ define a record type Ni such that (a) Ni contains one data item for each attribute

of Ri, and (b) for every key of Ri, define a key for Ni equal to the key of Ri.

2. For each key of Ri that appears as an attribute of Rj (a foreign key of Rj), define the set Lij between Ni and Nj (Ni owner and Nj member) as *optional* if the foreign key of Rj could be null, and *mandatory* if the foreign key of Rj cannot be null.

The transformation of a network schema to a relational schema is done similarly. Note that since CODASYL records do not have to have keys, we may have to use database keys instead. A database key is a system-added item to every record with unique values.

Query translation is done at compile time for compiled queries and at execution time for interactive queries. A compiled query is stored in the data dictionary for execution at a later time. There are two approaches for translating relational commands into network DML statements. The first approach is to translate relational commands one by one, as described in Vassilou and Lochovsky[11] and Zaniolo.[12] The second approach, used in this paper, is to translate a collection of relational commands as one optimization unit.

In the first translation approach each relational command is mapped into a set of DML commands, which have the same effect as the relational command, but on the network schema. This approach has the drawback of not taking full advantage of the optimization potential of the query being translated. An example can clearly illustrate this point. Consider the following relational schema and query:

SCHEMA:
DEPT(DNAME, HEAD, BUDGET)
STUDENT(SNAME, SSN, DEPT_NAME, SEX)
(DNAME and SNAME are keys, DEPT_NAME is a foreign key)

QUERY:
Print all information about all female students in departments with budgets less than $1,000,000.

The relational solution for this query could be:

```
T1←SELECT(STUDENT) where SEX = "female."
T2←SELECT(DEPT) where BUDGET < 1000000.
RESULT←JOIN (T1, T2)
where DNAME = DEPT_NAME.
```

The straightforward translation of these relational commands is a set of DML statements that searches all student records selecting only the female students, searches all department records selecting only the ones that have a budget smaller than $1,000,000, and joins the records from the two resulting sets if they have the same department name. Even though straightforward, this solution does not take into account the fact that there is a one-to-many relationship between relations DEPT and STUDENT. This relationship implies that a department can have many students, but each student belongs to only one department. Using the schema transformation rules previously explained, the relational schema is transformed into the network schema shown below.

$$\begin{tikzcd}
\boxed{\text{DEPT}} \arrow[d, "\text{DEPT\_STUDENT Set}"'] \\
\boxed{\text{STUDENT}}
\end{tikzcd}$$

Note that in this schema the relationship between the DEPT and STUDENT record types is explicitly shown as a set with DEPT record type being the owner and STUDENT record type as the member (set DEPT_STUDENT). Because of the availability of this set in a CODASYL database, we can combine the translation and optimization of the given query as follows:

1. Locate those department records that have BUDGET<1000000.
2. Search only the members of these departments, selecting the female students.

Note that this solution does not search the student records that are not members of departments with BUDGET<1000000. If the number of departments in the database is large, but only a few satisfy the stated condition, then the savings in the search time could be considerable.

Two important points should be noted in the above example. First, there is a specific pattern of relational commands in the query (i.e., SELECT, SELECT, JOIN). Second, there is a set of conditions among the variables (relations) used in the pattern of commands. For this example the conditions are that the DEPT record type must be the owner of a STUDENT record type and the join attribute is the attribute upon which the set type is defined. If either one of these criteria were not satisfied, we could not have optimized the query in this manner. The combination of a pattern and its associated set of conditions on the pattern variables are called a *template*.

One could find other templates of relational commands, similar to the template in the given example, that can be optimized and translated as a unit. These templates are defined at database design time, based upon the implementation of the local CODASYL databases. Optimized DML code for processing records that match each template is stored as the body of a subroutine, possibly in the data dictionary. The inputs to each subroutine are the relation and temporary relation names used in the commands and the output is a temporary relation that contains the results of each pattern. Having done this, the query translation and optimization procedure must look for patterns in the queries that match a template. Whenever it finds a match, it replaces the pattern with a temporary relation that represents the results of the optimized template. For each template found, a call to the subroutine corresponding to the template is generated in the code that will eventually be executed for the query. We call this code the query strategy table and store it in the data dictionary as well.

The process of looking for more templates is continued until either the query is reduced to a temporary relation or there are no more templates to be found. In the first case, the query strategy table contains all of the subroutine calls for execution of the query. In the second case, a one-by-one translation of the unmatched commands left in the query must also be integrated into the query strategy table. We describe how we define templates in our optimization scheme more fully in the next section.

To facilitate the template recognition phase of the optimization at compile time, the relational command list is transformed into a tree, called query tree. A query tree is a binary tree in which the nodes represent base and temporary relations and unary and binary operations on them. Examples of unary operations are SELECT, PROJECT, and UPDATE and examples of binary operations are JOIN, UNION, and INTERSECT. Figure 2 shows the query tree for the example query given above. In this figure DEPT and STUDENT are restricted to produce temporary relations T1 and T2, respectively. A JOIN of the two temporary relations T1 and T2 produces the results in RESULT.

LOCAL OPTIMIZATION

Our optimization approach emphasizes the matching of predefined patterns that can be processed as a unit on a CODASYL database. To illustrate this, we have defined a set of templates shown in Figures 3 to 5. The smallest of these templates consists of three nodes: two unary operations on R1 and R2 that are in a set and one binary operation on the results of the unary operations. In Figure 4, the only binary operations now specified for the templates are JOINs over the attributes that define sets. In this template, R1 is an owner/member of R2, which in turn is an owner/member of R3. Figure 5 shows the largest template defined. Larger and more complex templates can be defined. By increasing the number of templates defined on the local databases, a more thorough optimization can be done. However, the price of adding more templates is the additional storage cost of the patterns and the cost of matching these patterns to the query tree. We believe that a relatively small number of patterns will capture a high percentage of the optimization potential of query trees.

The code that executes a general template of *n* record types has the form:

```
For each r1 in R1 where P1 (r1)
  Get owner/member from R2
    For each r2 in R2 where P1 (r2)
      Get owner/member from R3
        .
          .
            .
              .
                b (r1, r2...rn)
                write ⟨results⟩
              .
            .
          .
        .
      end for.
end for.
```

Some Ri's have selection predicates, Pi, whose semantics can help access the Ri in an optimal way if Ri has a CALC key or is indexed. It is worthwhile to exploit the semantics of Pi in order to avoid unnecessary set walking. This is especially important in the case of R1 which is at the topmost level in the program and is, therefore, an entry point into the whole linked structure. The algorithm to determine the best available access path to r1 of R1 is as follows:

1. If the CODASYL schema information gives the access mode of R1 as CALC or INDEX and if the CALC key is a nonexistentially qualified attribute, ai, in P1, then determine whether R1 is accessible using the CALC key or index.[13] If R1 is accessible using the CALC key or index, then store this information in an access strategy structure for R1 for use at execution time.

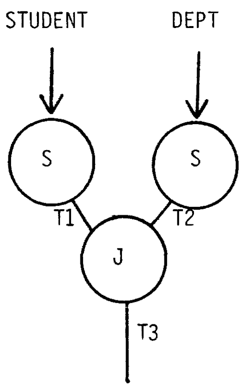
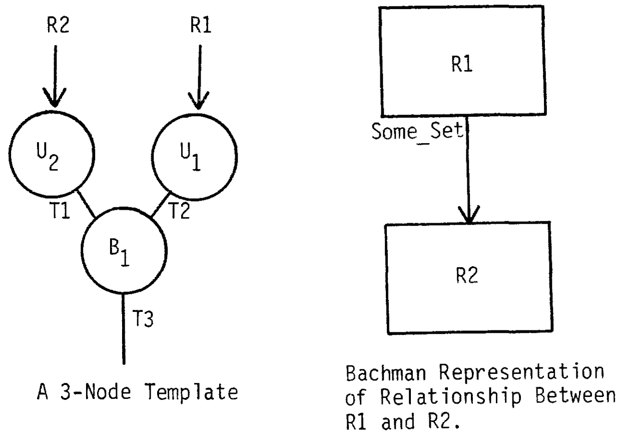


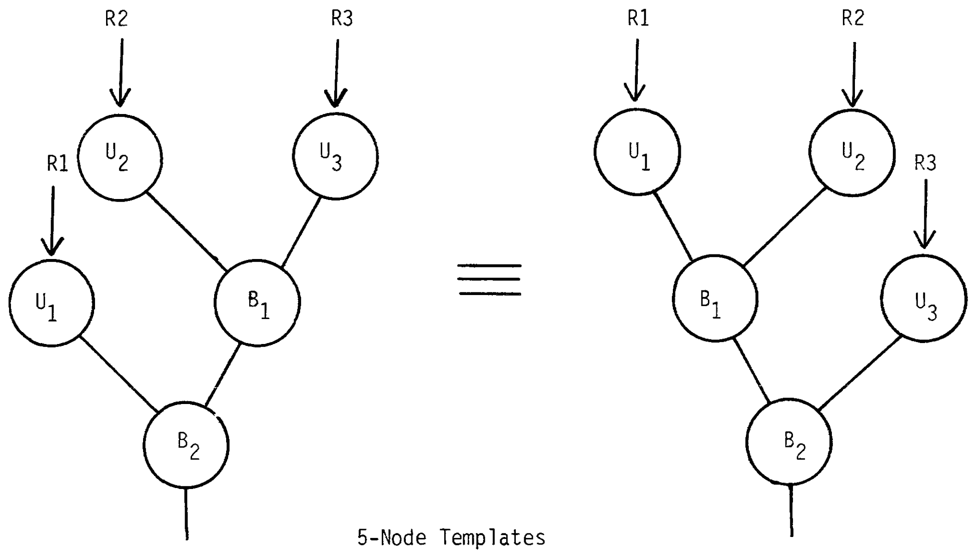
Figure 2—STUDENT-DEPARTMENT query tree

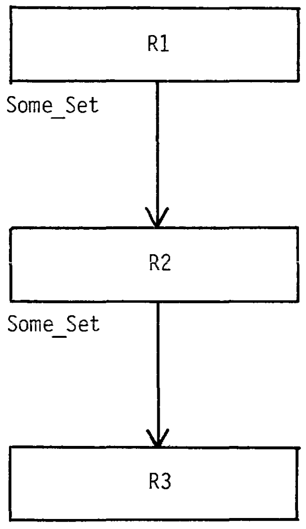


Legend:

S	Select
J	Join
Ui	Unary Operation
Bi	Binary Operation
Ri	Record_type/Relation
Ti	Temporary Results

Figure 3—3-node example





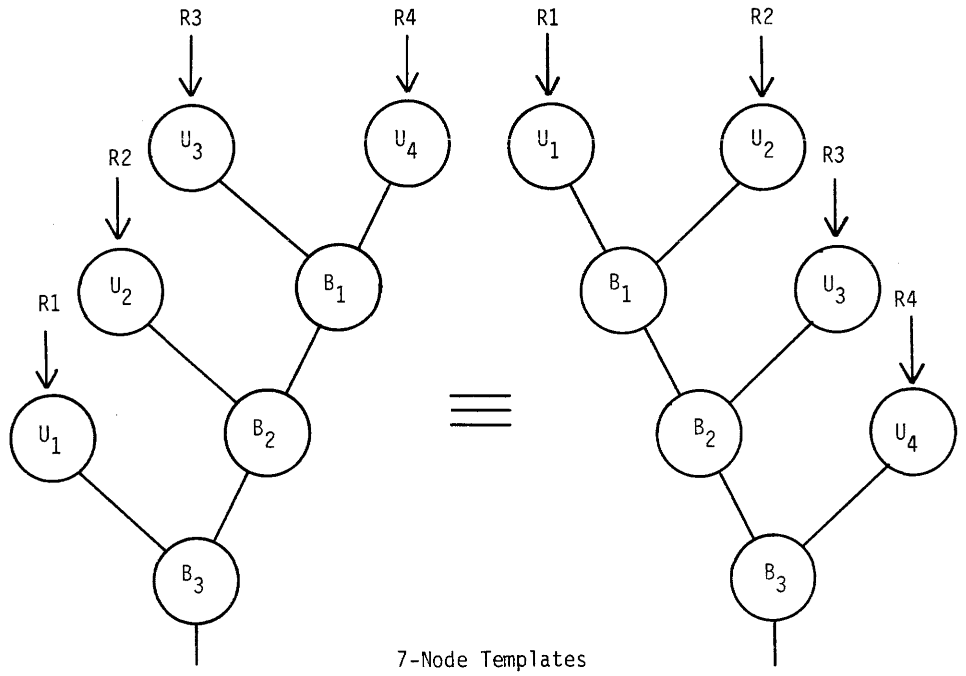
Bachman Diagram of the R1_R2_R3 Relationship

Figure 4—5-node example

2. Else, if the access mode of R1 is the VIA SET mode and if the CALC key or index of an owner of R1 is also a nonexistentially qualified attribute, ai, in P1, then determine, as in step 1, if the owner of R1 is accessible using the CALC key or index. If true, then store this information concerning R1's owner in a strategy table for R1. The r1's will be located at execution time by first hashing or indexing to the owner and then retrieving the r1's.
3. Else, each r1 must be retrieved by sequential access across the realms where R1 resides.

An algorithm for pattern matching is used to match subtrees of the query tree against the predefined templates. In version 0 of DDTS, we have employed a finite state automaton to recognize these optimizable subtrees.[15] The algorithm to generate an optimized query execution strategy is as follows:

1. Use the finite state automaton to match the largest possible pattern contained in the query tree.
2. Generate a call to the appropriate precompiled routine that processes this pattern and store that call in the query strategy table. The input parameters are R1...Rn of the pattern plus the Pi's.
3. Replace the recognized pattern with a dummy node (i.e., a SELECT ALL node on resulting temporary relation).



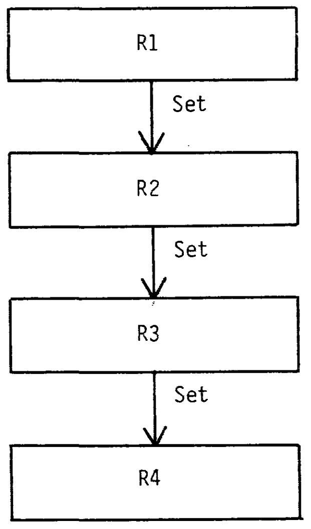


Figure 5—7-node example

4. Repeat steps 1 to 4 on the reduced tree until no more patterns can be matched.
5. Step through the remaining query tree node by node and translate each relational operation into an operation on the network database.[11] These operations are integrated into the code that calls the predefined routines that process the patterns.

IMPLEMENTATION

We mentioned earlier that the extended relational model of data is used to describe both the global representation schemas and the local representation schemas. Tuples of relations in these schemas are uniquely identified by system-assigned values in surrogate attributes. These surrogate keys also define the CODASYL sets of the local internal schema in DDTS. During the translation of groups of user queries to their equivalent relational algebra operations, we found that approximately 95 percent of the JOIN operations are over these surrogate keys. This is because the query language at the external schema level in DDTS is a graph-oriented language, GORDAS.[4] Data selection in GORDAS is influenced by the dependencies among entities. An example will illustrate the processing of a transaction or subpart of a transaction in DDTS.

A school library maintains records of overdue books in the following way: Students are grouped according to their departments; each student's overdue collections are recorded against his or her name. This information is modeled by using the extended relational schema as the conceptual model. The local database is, however, a CODASYL network database. Queries are expressed in relational algebra and must be translated and optimized to operate on the database. Here we process a user request to find and list all overdue books (along with the names of the defaulting students) held by graduating seniors (i.e., SEMESTER = 8) in the computer science department.

The conceptual schema is given below. Note that DEPT#, S#, and B# are the surrogate attributes of DEPARTMENT, STUDENT, and OD_BOOKS relations, respectively.

DEPARTMENT (DEPT#, DEPTNAME, DEPTHEAD)
STUDENT (S#, SNAME, DEPT#, SEMESTER)
OD_BOOKS (B#, S#, BNAME)

The transaction in SEQUEL-like form would be:

T1 ← Select DEPARTMENT where DEPTNAME = "Computer Science".
T2 ← Select STUDENT where SEMESTER = 8.
T3 ← Join T1 and T2 over DEPT#.
T4 ← Join OD_BOOKS and T3 over S#.
T5 ← Project T4 over SNAME, BNAME.

Figure 6 shows the query tree for this user request. Note that the initial selection on OD_BOOKS is a trivial SELECT ALL operation, thus, no temporary needs to be formed. Figure 7 is a Bachman diagram of the CODASYL version of the local internal schema. Up to the final PROJECT operation the query matches the template found in Figure 4. The pattern can be optimized as a unit. Let us assume that the DEPARTMENT record type is calced on DEPTNAME and that the other record types have the VIA SET location mode. In addition we assume the following functions. Function "concat" concatenates records. Function "retain" retains only the named fields and therefore is equivalent to the relational project. Function "write" is self-explanatory. The functions "get_first_member" and "get_next_member" return the first record occurrence and subsequent record occurrences, respectively. The "get_first" function is used to retrieve the DEPARTMENT record(s) and returns them either by sequential scan or by using the CALC or index keys.

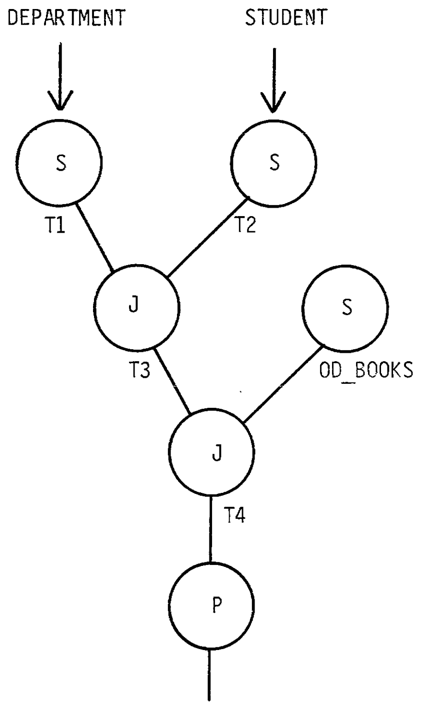


Figure 6—Example query tree

The optimized query strategy is generated as follows:

```
r1 = get_first (DEPARTMENT)
    while (r1 exists) do
        if (r1.DEPTNAME = "Computer Science") then
            r2 = get_first_member (STUDENT)
            while (r2 exists) do
                if (r2.SEMESTER = 8) then
                    r3 = get_first_member (OD_BOOKS)
                    while (r3 exists) do
                        TEMP = concat(r1,r2,r3)
                        TEMP = retain(BNAME,
                          SNAME).
                        write ⟨TEMP⟩
                        r3 = get_next_member
                          (OD_BOOKS)
                    end while
                end if
                r2 = get_next_member (STUDENT)
            end while
        end if
        r1 = get_next_member (DEPARTMENT)
    end while
```

PERFORMANCE ANALYSIS

We will analyze the effectiveness of the local optimization techniques described in this paper by monitoring the performance of DDTS. We have designed a detailed study wherein two parameters will be varied. The local optimization can

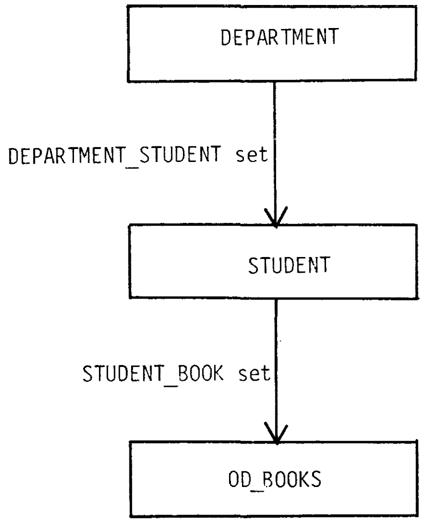


Figure 7—A Bachman diagram of the relationship between DEPARTMENT-STUDENT-BOOKS

be turned on and off by a software switch at each DBMS in the system. If no local optimization is performed, then a straightforward relational operation to network operation(s) translation will be performed. The second parameter will be the selection of a query stream to run on the system. By defining different query streams as to their primary processing purpose, we will be able to measure the effect of local optimization on retrieval intensive queries, update intensive queries, insertion/delete queries, and various query mixes. The results of this study will provide valuable information for improving future versions of the local translation and optimization module.

SUMMARY AND FUTURE EXTENSIONS

We have described a general method of translating and optimizing a database query, expressed in relational algebra, to network DML commands. Our optimization approach rapidly recognizes the query access patterns that are potentially beneficial for optimization. This is in contrast to other proposed schemes that translate each individual relational command without regard to the interrelationship of the commands and the variables used in these commands. The algorithm has been implemented as the local query optimization and translation of Honeywell's distributed database testbed system, DDTS.

Our future plans call for

1. Integrating other data models and local DBMSs into DDTS: Our methods of translation and optimization can readily be extended to handle other systems with a network model, systems with a hierarchical data model, and systems based on the relational data model. Pattern matching and predefined access path recognition will remain the critical features of the methodology.
2. Sharing local optimization information on the network: Data access paths may be reorganized and enhanced (e.g., by adding new indexes) at a local DBMS. This information should be broadcast on the network. The nodes that handle the distributed query optimization can then send subqueries to the most effective nodes for local translation and optimization.
3. Designing better methods of handling procedural constraints in queries: Pattern matching is difficult when decision structures are included in a query. In the third section, we proposed a simple, but less than satisfactory, method of dealing with this problem. Any pattern that could not be directly matched was translated operation by operation for processing in the local DBMS. We are studying ways to make pattern matching more general in future versions of DDTS.
4. Using a more powerful pattern recognition algorithm to match patterns: The patterns can be matched using a recursive procedure that will recognize any pattern of any complexity. So, in the future, such a recursive algorithm will replace the simple finite state automaton we have used. This means that the pattern recognition algorithm will also generate code for processing any pattern recognized.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under Grant #ECS-8007683.

REFERENCES

1. Tsichritzis, D., and A. Klug (eds.). "The ANSI/X3/SPARC DBMS Framework Report of the Study Group on Data Base Management Systems." AFIPS Press, Montvale, N.J.: AFIPS Press, 1977.
2. Devor, C., and J. Weeldreyer. "DDTS: A Testbed for Distributed Database Research." Technical Report HR-80-268, Honeywell CCSC, Bloomington, Minnesota, August 1980.
3. Wong, E. "The Design of Representation Schemas." Technical Report HR-80-265, Honeywell CCSC, Bloomington, Minnesota, July 1980.
4. Elmasri, R. "GORDAS: A Data Definition, Query and Update Language for the Entity-Category-Relationship Model of Data." Technical Report HR-81-250, Honeywell CCSC, Bloomington, Minnesota, January 1981.
5. Hevner, A. "Transaction Optimization Techniques in a Distributed Database System." Technical Report HR-81-259, Honeywell CCSC, Bloomington, Minnesota, June 1981.
6. Devor, C. "Experience with Distributed System Design in DDTS." *Proceedings of IEEE Third International Conference on Distributed Computing Systems,* Fort Lauderdale, Florida, October 1982.
7. Elmasri, R., Devor, C., and Rahimi, S. "Notes on DDTS: An Apparatus for Experimental Research in Distributed Database Systems." *ACM SIGMOD Record,* July 1981.
8. Codd, E. "Extending the Relational Model to Capture More Meaning." *ACM Transactions on Database Systems,* Volume 4, Number 4, December 1979, pp. 397–434.
9. Dayal, U. and N. Goodman. "Query Optimization for CODASYL Database Systems." *Proceedings of the International Conference on the Management of Data,* ACM SIGMOD 1982, Orlando, Florida, pp. 138–150.
10. Honeywell Information Systems. *Integrated Data Store (IDS) Reference Manual,* Wellesley, Massachusetts, 1972.
11. Vassiliou, Y., and F. Lochovsky. "DBMS Transaction Translation." *Proceedings of IEEE COMPSAC '80,* Chicago, 1980, pp. 89–96.

12. Zaniolo, C. "Design of Relational Views Over Network Schemas," *Proceedings of the International Conference on the Management of Data,* ACM SIGMOD '79, Boston, June 1979.
13. Astrahan, M., and D. Chamberlain. "Implementation of a Structured English Query Language." *CACM,* Volume 18, Number 10, 1975.
14. Committee on Data Systems Languages. CODASYL Data Base Task Group Revised Report, ACM, 1978.
15. Hoffman, C., and J. O'Donnell. "Pattern Matching in Trees," *JACM,* Vol. 29, No. 1, January 1982.

Progress towards database management standards

by DONALD R. DEUTSCH
General Electric Information Services Co.
Nashville, Tennessee

ABSTRACT

The first proposals for database management standards appeared in the late 1960s. Work began on a U.S. national standard in 1978. Today there are no domestic or international database management standards, although organizations throughout the world are working toward this goal. This paper describes these various organizations and the current status of their work. It outlines recent changes in the structure and scope of American database management standardization activities that have substantially improved the outlook for timely results.

INTRODUCTION

Information processing professionals have been anticipating database management standards for over a decade. In the late 1960s and early 1970s, a flurry of reports surveyed and analyzed features of then extant database management systems (DBMS), detailed requirements for future DBMS products, and proposed specific languages for describing and manipulating databases.[1,2,3,4] While the Conference on Data Systems Languages (CODASYL) candidate for a standard database management language interface was evolving, the next decade saw a proliferation of DBMS approaches, products, and applications. This increasing diversity and pervasiveness of database management software motivated renewed interest in standard approaches for database management.[5]

Today, there are no domestic or international database management standards. Work that began in 1978 on the first American database management standard is still ongoing; but recent changes in the way DBMS standards are being developed, in their scope and in their relationships with other standards, have substantially improved the outlook for database management standardization. This paper describes the current state of DBMS standardization and updates a 1980 publication describing the various organizations addressing DBMS standards.[6]

STANDARDIZATION BODIES

Standards for database management software, like those for other information processing components, are developed in a vast world-wide environment that includes thousands of individuals working within many organizations. The following paragraphs briefly describe the groups and committees most instrumental in developing database management standards today and the current status of their work. A more complete discussion of how EDP standards of all types are developed appears in Prigge.[7]

International Standards Organizations

The International Standards Organization (ISO) develops standards to facilitate the international exchange of goods and services and to promote intellectual, scientific, technological, and economic cooperation. International standards are increasingly essential for world-wide trade; in the absence of international standards, differing national technical requirements rival trade tariffs as barriers to international commerce. ISO member bodies are responsible for standardization in their respective countries; for instance, the United States is represented by the American National Standards Institute (ANSI). ISO work is carried out in technical committees composed of interested member bodies with one member body serving as secretariat. The United States holds the secretariat for ISO technical committee TC 97—Computers and Information Processing.

Three groups working within the SC 5 Programming Languages subcommittee of TC 97 are currently addressing database management issues. The Conceptual Schema Working Group, WG 3, issued a report on concepts and terminology in April 1982.[8] The DBMS Coordination Working Group is charged with evaluating, planning, and coordinating future TC 97 efforts in the area of database management. Finally, an international database experts group was convened in December 1982 to advise the ANSI X3H2 Database Committee on international standardization issues. In addition to these three bodies primarily concerned with database management, experts groups for COBOL and FORTRAN advise their ANSI counterparts on the development of international standards, including facilities for using future ISO database language standards.

National Organizations

To participate properly in international standardization activities and to develop consensus domestic standards, national bodies must provide mechanisms for coordinating diverse interests within their respective countries. Although the basic responsibilities of national standardization bodies are similar from country to country, their organizational structures and levels of participation for government, industrial, and consumer interests vary widely. Three organizations frequently are associated with database management standardization in the United States; each has a distinct role in developing DBMS standards.

CODASYL

The Conference on Data Systems Languages (CODASYL) is dedicated to the development of computer languages independent of specific hardware characteristics. Participation in CODASYL committees is not limited by nationality, but CODASYL's COBOL and network database management specifications have been the basis of major standardization efforts within the United States. CODASYL, a developmental body that produces language specifications, is *not* directly involved in establishing standards. Each CODASYL committee periodically publishes the results of its language development efforts in a *Journal of Development* (JOD). Other organizations can use JOD specifications as a basis for implementation and standardization efforts.

CODASYL bodies with database management responsibilities include the Data Description Language Committee (DDLC), the COBOL committee, and the FORTRAN Database Language Committee (FDBLC). DDLC and COBOL last published their JOD's in 1981. Neither group has actively addressed database management standardization issues in recent months. FDBLC published its second JOD in January 1980; it has been dormant since.

ANSI

The American National Standards Institute (ANSI) is a federation of more than 180 organizations representing trade, professional, commercial, labor, and consumer interests. ANSI is the official representative of the United States in international standardization efforts, and in its capacity as secretariat it directs the work of many ISO technical committees. The Computer and Business Equipment Manufacturers Association (CBEMA) holds the secretariat for the ANSI X3 Committee on Computers and Information Processing. Work is carried out within X3 by standing and technical committees. Standing committees advise X3 on the administration, evaluation, allocation, and scheduling of standards projects. Technical committees are charged with developing draft standards on assigned topics; members representing a wide range of organizations are selected based on their individual technical expertise.

Four technical committees and an advisory body within X3 are working on database management. They are X3H2, X3H4, X3J3, X3J4, and the SPARC Database Systems Study Group.

X3H2. The X3H2 Database Committee is charged with developing American national standards (ANS) for database management facilities based on the network and relational data models, including both data definition languages and generic operations on DDL-defined structures. The X3H2 network database language (NDL) specification is derived from the CODASYL COBOL and DDLC JOD's of January 1978. X3H2 adopted a formal specification for SQL as its base document for a relational database language (RDL) standardization effort initiated in October 1982. Originally charged with developing a draft standard for a network data description language (with access languages to be provided by the COBOL and FORTRAN committees), X3H2's charter was expanded in the past 18 months to include generic operations and the relational model. This broadening of scope and elimination of dependency on multiple technical committees greatly improves the prognosis for an early ANS database language.

X3H4. Established in 1980, this committee is charged with developing an ANSI standard for an information resource dictionary system (IRDS). To date, X3H4 has produced a dynamic requirements document and a skeletal functional specification for an IRDS standard. It is working to evolve a draft proposed standard for review by X3 within the next year.

X3J3. This committee is developing a revision for the current ANSI FORTRAN standard. The X3J3.1 Database Task Group, established in 1979 to consider DML and subschema languages for the next FORTRAN standard, has not met in more than two years. This recent lack of progress in the database area reflects X3J3's belief that it is premature to consider incorporation of specific database capabilities in FORTRAN. However, recognition of the potential impact from the incorporation of database functionality in FORTRAN motivated much of the restructuring and enhancing of the language for the revised standard currently being developed.

X3J4. The COBOL committee is working to revise the current ANSI standard; COBOL 198X will not include subschema and data manipulation language (DML) facilities as once planned. Instead, X3J4 is considering X3H2 database specifications for possible inclusion in future revisions of ANS COBOL. A newly established task group, X3J4.1, is responsible for defining the COBOL syntax to interface with the X3H2 NDL. X3J4 will not delay release of a revised standard until mechanisms have been defined for integrating the X3H2 database facility into COBOL. The recent decoupling of programming language syntax from DBMS operators (now being defined generically by X3H2) means that interfaces to COBOL and other programming languages can be specified whenever programming language committees so desire; the timely availability of a database language standard is not dependent on their action.

Database Systems Study Group. Acting as an advisor to the X3 Standards Planning and Requirements Committee (SPARC), the Database Systems Study Group (DBSSG) is charged with planning and coordinating future ANSI database standardization efforts. Prior to 1980, X3 was considering standards for CODASYL-network database languages only; X3H2 was charged with developing a DDL, with X3J3 and X3J4 responsible for defining DML and subschema languages for FORTRAN and COBOL, respectively. In the past three years, ANSI restructured and broadened its database standards development effort in response to DBSSG suggestions. Today, ANSI is addressing relational database management as well as the CODASYL-network approach. X3 has clearly defined and separated responsibilities for programming language syntax and generic DBMS operators. A single committee, X3H2, has responsibility for defining both structure definition languages and operations on those structures, while programming language committees are charged with defining the specific syntax for accessing databases. Finally, data dictionary/directory standardization is well underway within X3H4.

United States Government

The U.S. Government depends on standards for competitive procurement of computer hardware, software, and services. The Institute for Computer Sciences and Technology (ICST) of the National Bureau of Standards (NBS) develops federal standards and guidelines for effectively using computers in the government. Federal Information Processing Standards (FIPS) cover all aspects of computer use; FIPS standards and guidelines issued by NBS apply to procurement and management practices of federal agencies. To facilitate development of industry standards for database management systems that can be applied to government problems, NBS participates along with other federal agencies in many of the database standardization bodies previously described.

Other Organizations

The European Computer Manufacturers Association (ECMA) is a nonprofit association of computer manufacturers whose purpose is to develop standards for European data-processing systems. Over the past several years, ECMA's database technical committee, TC22, has actively followed the work of CODASYL and ANSI, publishing constructive critiques and enumerating incompatibilities among related specifications.

The International Federation for Information Processing (IFIP) is not a standards development body per se, but it sponsors studies that can form the basis for international standards. IFIP is a federation of technical societies concerned with information processing. More than 30 countries are represented in IFIP. The American Federation of Information Processing Societies (AFIPS) represents the United States; the ACM, the IEEE, and other professional organizations are members of AFIPS.

CONCLUSION

Although there are no existing domestic or international database management standards, recent changes in the structure and scope of American DBMS standardization activities have improved the prognosis for timely results. The preceding paragraphs briefly surveyed the international and American groups most often associated with DBMS standardization.

REFERENCES

1. CODASYL Systems Committee. *A Survey of Generalized Database Management Systems,* Association for Computing Machinery, May 1969.
2. *CODASYL Database Task Group Report,* Association for Computing Machinery, April 1971.
3. CODASYL Systems Committee. *Feature Analysis of Generalized Database Management Systems,* Association for Computing Machinery, May 1971.
4. Sibley, Edgar H. "Standardization and Database Systems." *Proceedings of the Third International Conference on Very Large Databases,* IEEE and ACM, October 1977.
5. Joint GUIDE-SHARE Database Requirements Group. *Database Management System Requirements,* November 1970.
6. Deutsch, D. R., and E. K. Clemons. "DBMS Standards: Current Status and Future Directions." *Proceedings of the Sixth International Conference on Very Large Databases,* IEEE, ACM, and CIPS, October 1980, Long Beach, Cal.: IEEE, pp. 431–433.
7. Prigge, R. D. et al. *The World of EDP Standards.* Blue Bell, Pa.: Sperry Univac, 1978.
8. Van Grithuysen, J. J., ed. *Concepts and Terminology for the Conceptual Schema and the Information Base.* New York: ISO/TC97/SC5/WG3, April 1982.

Command use in a relational database system

by JOHN D. JOYCE and DAVID R. WARN
General Motors Research Laboratories
Warren, Michigan

ABSTRACT

A study of commands in a relational database system was undertaken to provide a basis for improving future implementations of relational techniques. The use of Regis (Relational General Information System), an interactive relational database system developed at General Motors Research Laboratories, was monitored to accumulate a large amount of data about distribution of command uses across a variety of applications developed by users with a diverse set of capabilities. Of the basic relational commands, it is important that searching, PROJECTION, and JOIN operations be efficient. However, optimizing INTERSECTION and "exclusive or" may not warrant the time and effort it would require. Although proportions of use varied, the rankings of use of relational operations were reasonably independent of applications and the programming expertise level of the users.

INTRODUCTION

An interactive relational system has been measured to determine relative use of different kinds of commands by industrial users. This information was gathered to gain some insight into where emphasis should be placed for efficient implementation of future relational systems, either in hardware or software. Data have been acquired over a 6-month period by monitoring all completed sessions on one computer system.

Descriptions of the salient characteristics of the system, the command language, the monitoring facilities, and the major applications are presented in following sections. The characteristics of command use will be presented by command categories and by application area.

REGIS DATABASE SYSTEM

Regis (Relational General Information System[1]) is an interactive relational database system developed at the General Motors Research Laboratories. Since 1975 it has been in production use by many GM divisions and staffs. The system is designed to provide convenient, powerful, and flexible information manipulation facilities for information storage, retrieval, and analysis. All data are represented in a simple tabular form to provide an easily understandable and easily manipulated view of data. Graphical display facilities to plot the data and statistical functions to analyze the data are also incorporated in the package.

Regis is designed for the interactive user who asks questions, receives answers, and then asks further questions based on the answers. It is particularly well suited to applications in which the queries cannot be defined in advance. Regis, however, also provides the ability to package commands in parameterized command files with sufficient logic control commands to permit the development of sophisticated applications.

HIGHLIGHTS OF THE COMMAND LANGUAGE

Regis uses an interpreted algebraic command language. The command name appears as the first word of each command except when the results of the command are stored in a table different from the source table. In that case the command name is preceded by "table_name = ".

There are in total 66 commands. These have been grouped into six major categories for analysis in this paper.

Basic Relational Commands

Eight basic relational commands were selected as the focus for analysis in this paper because they correspond most closely to the relational operations described in the literature.[2] This group consists of SUBSET, PROJECTION, JOIN, JOINALL, UNION, INTERSECTION, DIFFERENCE, and XOR. An explanation of how these commands relate to, or differ from, relational terminology will be given.

The SUBSET command selects rows from a table using a boolean expression of column relationships to be satisfied by the selected rows. An extensive set of pattern-matching functions is provided for text columns in addition to the typical functions for numeric relationships. The selected rows are placed in a result table, which has the same form (columns) as the source table. The SUBSET command is sometimes referred to in the literature as "selection" or "restriction." SUBSET is a "search" command.

The PROJECTION command selects specified columns from a table. There are options for reordering of columns, sorting of rows and eliminating duplicate rows.

JOIN and JOINALL combine columns from two source tables into a new target table where values match in specified columns. The resulting table will contain all columns from both source tables, except that the columns being matched occur only once. There are a number of variations in terminology in the literature describing various facets of the join capability. The Regis implementation of JOIN is usually referred to as a *natural join*.[2] A minor extension is that the named columns (domains or attributes) that are being matched to perform the join operation in the two source tables need not have the same names, although the data types must match. The *theta-join* operations (where theta can be any of the comparison operators $=, >, <, \ldots$) are not supported, except for $=$. M-way joins where $M > 2$ are also not supported directly.[3] Users can, of course, repeatedly apply a join operator to the results of a previous "join" to simulate an M-way join. Neither the users nor the implementors during seven years of use have perceived much need for these kinds of additional primitive operations. M-way joins would be a convenience at times, however.

A different extension of join, however, was often requested by users. The JOINALL command is an implementation of the generalized symmetric natural join as defined by Lacroix and Pirotte.[4] It differs from this natural join in cases where a match does not exist on the columns being joined; the target row (tuple) will be augmented by null values for the columns of the appropriate source table.

DIFFERENCE, INTERSECTION, UNION, and XOR (exclusive or, i.e., symmetric difference) perform the normal set operations on two tables. The Regis implementation permits the comparisons to be made on the entire row (all columns) or on selected columns whose values are to be matched. UNION and XOR require both tables to have the same form for all columns. DIFFERENCE and INTER-

SECTION always take rows from the first source table only so that the second source table has only to have the same data types for the columns whose values are to be matched. There can be any number or type of additional columns in the second source table.

Logic Control Commands

The logic control commands provide the ability to package Regis commands in a command file with optional parameters. The command procedures can be used to implement new functions or to provide a complete application system. The commands include conditional execution, repeated execution of a group of commands, parameter definition, and local variables. An extensive set of built-in functions complements these commands and provides string manipulation, access to table, column, and row information, arithmetic operations, and so on.

Analysis and Modification Commands

A variety of commands are provided to analyze and modify tables. Table modification functions include sorting a table on one or more columns, performing arithmetic on columns, replacing particular values in a column, appending one table to another, and transposing a table to interchange rows and columns. Analysis functions are available to generate statistics such as minimum, maximum, average, count, sum, and standard deviation for numerical columns of a table. These statistics can be obtained for all rows of a table or for groups of rows within a table where each group has equal values in one or more selected columns. Curves for trend prediction can be fitted to data using polynomial or least-squares fit. Other commands are available to perform multivariate regression analysis and to generate bivariate multiple correlation coefficients.

Table and File-Handling Commands

The table-handling commands provide the ability to create, delete, or rename tables or columns and to add or delete rows or columns. File functions include reading, writing, releasing and backing up files.

Printed and Graphical-Output Commands

A variety of output commands are provided for both printed and graphical formats. Printed output ranges from a simple listing of a table at the terminal to a completely formatted report. Simple bar charts and plots may also be printed. Graphical output is produced through a tightly coupled interface to SIMON (a GMR-developed interactive graph-plotting package) and a command interface with data transfer by a disk file to a commercial interactive package. These packages can be used to produce high-quality graphs from data contained in Regis tables. Other output commands display messages to the terminal, clear the terminal display and control pagination of printed output.

Miscellaneous Commands

Other commands are provided to display debugging information about tables and files, set Regis default values, invoke DO-IT Menu System,[5] establish command synonyms, and terminate a Regis session. A HELP command will display the syntax for any command, and a NEWS command will display announcements for the users. An OBEY command can be used to issue operating system commands without leaving Regis.

INSTRUMENTATION CAPABILITIES

Monitoring capabilities had previously been incorporated in Regis to provide information on every Regis session.[6] A 1-row table is created to describe a session: the user ID, total commands, total central processing unit (cpu) time, total elapsed time, and so on. For every session that terminates normally, this table is written to a file that is periodically analyzed to provide a global summary of Regis use.

For purposes of the current study, the command interpreter was modified to maintain an array containing the number of times each command is executed during a session. At the end of each session the array of command counts was put into a table and written out along with the session data. These data were collected for all Regis sessions on one computer over a 6-month period.

CHARACTERISTICS OF INTERACTIVE USAGE

As background for the study on relational command use, some characteristics of the interactive sessions will be outlined. The distribution of the durations of the on-line interactive sessions is shown in Figure 1. There is a high proportion

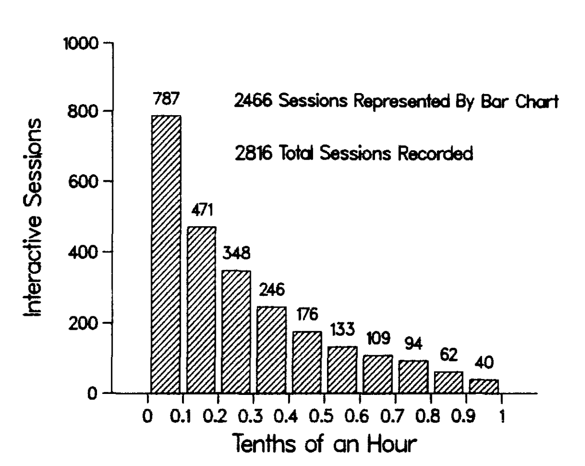


Figure 1—Distribution of durations of interactive sessions

of sessions in which the total duration is much shorter than expected. We theorize that either there are numerous short sessions for the purposes of updating databases with small amounts of data, or there are numerous requests for small amounts of information.

The number of commands used per session varies greatly because commands are used in two basic modes. One mode is to key in each command at an interactive terminal for immediate execution. In this mode a relatively small number of commands are issued in an interactive session. In the second mode commands are executed from command files. Some user applications rely heavily on prepackaged command files. Between the two extremes, there is a fairly smooth distribution. (See Figure 2.) To some extent this gives an indication of the degree to which ad hoc and preplanned procedures are used.

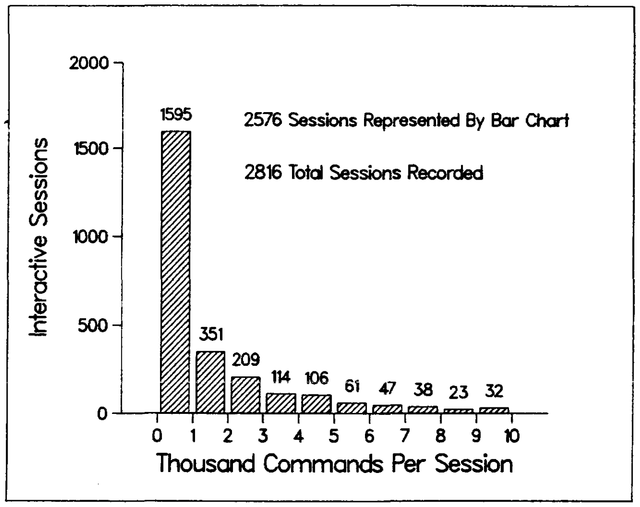


Figure 2—Distribution of number of commands per session

USE OF BASIC RELATIONAL COMMANDS

Use Across All Applications

Regis applications come from a variety of areas, including workload scheduling, engineering tests, quality control, machine failure analysis, and analysis of business operations. For most long-term applications, users have developed extensive collections of command files to handle routine database functions and produce standard reports. In addition, two major functions of Regis have been implemented in command files. A composite analysis of all command executions will be presented first.

The data in Figure 3 include all the sessions that were monitored beginning December 1980 through May 1981. The TOTALS value for the number of sessions column in Figure 3 is the total number of sessions that were monitored; it is not the sum of the number-of-sessions entries tabulated for each command. Clearly, searching of tables (SUBSET) is the most heavily used relational function. PROJECTION and JOIN are also very important in the total of uses. UNION, DIFFERENCE, and JOINALL are used considerably less often, but are still important capabilities for a number of applications. Both the XOR command and the INTERSECTION command could reasonably be implemented as command files of UNION or DIFFERENCE commands.

Relational Command Usage in Two Regis Extensions

The next analysis focuses on relational commands used in providing two major extensions to Regis, the Regis Report Generator and the DO-IT Menu System.[5] Both are implemented with Regis command files. For that reason, the use of either facility could potentially distort the interactive characteristics and distribution of command uses for a session, because many commands are executed for each command that is issued by a user.

The Regis Report Generator provides controls to format reports from Regis tables for printing or display at a terminal. This facility has proven to be very popular with users for preparing reports suitable for presentation or publication. The DO-IT Menu System provides a menu interface to almost all Regis commands. Instead of having to learn or remember the syntax of commands, users make selections from a menu. The capabilities of the system are evident in the choices displayed and there is far less typing involved. A custom-menu facility provides the ability to create, modify, and execute specialized menus adapted to the needs and terminology of an application.

Regis commands were used to implement these extensions because it took much less effort than it would to use a conventional programming language such as PL/I. Sessions that used either of these extensions were separated for analysis to determine whether their characteristics or the programming experience of the implementors would make any significant differences in the uses distribution of relational commands. User application commands may also be present in these sessions, but it is assumed that the two major Regis extensions dominate these data.

The distribution of uses of basic relational commands for these two extensions is quite similar to the distribution for all

COMMAND NAME	NUMBER OF SESSIONS	NUMBER OF COMMANDS	PERCENT OF RELATIONAL COMMANDS
SUBSET	1 901	182 213	72.4
PROJECTION	1 623	38 990	15.5
JOIN	836	18 601	7.4
UNION	568	5 460	2.2
DIFFERENCE	477	3 498	1.4
JOINALL	190	2 418	1.0
INTERSECTION	27	275	0.1
XOR	1	1	0.0
TOTALS	2816	251 456	100.0

TOTAL NUMBER OF ALL COMMANDS FOR THESE SESSIONS = 14 094 675

Figure 3—Basic relational commands sorted by usage

COMMAND NAME	NUMBER OF SESSIONS	NUMBER OF COMMANDS	PERCENT OF RELATIONAL COMMANDS
SUBSET	839	54 098	58.2
PROJECTION	797	24 075	25.9
JOIN	507	6 937	7.4
UNION	472	5 327	5.7
DIFFERENCE	356	1 827	2.0
JOINALL	65	585	0.6
INTERSECTION	13	140	0.2
XOR	0	0	0.0
TOTALS	843	92 989	100.0

TOTAL NUMBER OF ALL COMMANDS FOR THESE SESSIONS = 7 214 195

Figure 4—Sessions using DO-IT menu and report generator

applications. The order of frequency of use of relational commands is identical. See Figure 4.

Analysis of User Applications

The next analysis includes only the sessions that did not have any DO-IT menu or Regis report generator commands. These results are shown in Figure 5. These data would seem to indicate that users in normal user applications do more searching of tables than is done in the Regis extensions and that fewer PROJECTION operations are done. However, subsequent examination of some of the major applications shows that there is sufficient variability in the use of relational commands that this conclusion would not be valid. One can also observe that both the JOIN and JOINALL operations are used much more than any of the INTERSECTION, UNION, or XOR operations. For these user applications, the latter three commands are little used. Thus, nonoptimized implementations of these three commands would be tolerable.

Four of the largest user applications have also been summarized. For each application, there is a group of anywhere from 5 to 15 users so that no application represents the characteristics of any individual. Application A is an application run by manufacturing people who are tracking quality control for the manufacture of small parts. Application B is the monitoring of trends of failures in major automotive assemblies in another manufacturing division. Application C is the analysis of sales forecasts to make plans periodically for future capital, tooling, space, and other plant requirements for component manufacturing. Application D consists of monitoring project schedules, computing effects on workloads when plans for new project target dates are shifted, and minimizing overtime of a variety of skilled people working on various facets of these projects.

COMMAND NAME	NUMBER OF SESSIONS	NUMBER OF COMMANDS	PERCENT OF RELATIONAL COMMANDS
SUBSET	1062	128 115	80.8
PROJECTION	826	14 915	9.4
JOIN	329	11 664	7.4
JOINALL	125	1 833	1.2
DIFFERENCE	121	1 671	1.0
INTERSECTION	14	135	0.1
UNION	96	133	0.1
XOR	1	1	0.0
TOTALS	1973	158 467	100.0

TOTAL NUMBER OF ALL COMMANDS FOR THESE SESSIONS = 6 880 480

Figure 5—Relational commands in user applications

COMMAND NAME	ALL APPLI-CATIONS %	MENU, REPORT EXTEN-SIONS %	APPLI-CATION A %	APPLI-CATION B %	APPLI-CATION C %	APPLI-CATION D %
SUBSET	72.4	58.2	49.6	91.7	60.3	51.6
PROJECTION	15.5	25.9	28.7	1.1	34.9	28.8
JOIN	7.4	7.4	12.1	7.1	2.0	8.4
UNION	2.2	5.7	.0	.0	.4	8.5
DIFFERENCE	1.4	2.0	4.0	.1	1.5	2.7
JOINALL	1.0	.6	5.0	.0	.7	.0
INTERSECTION	.1	.2	.6	.0	.2	.0
XOR	.0	.0	.0	.0	.0	.0
TOTAL RELATIONAL COMMANDS	251,456	92,989	38,594	115,948	9,959	61,492
TOTAL OF ALL COMMANDS	14,094,675	7,214,195	401,910	950,192	1,193,309	3,328,123
NUMBER OF SESSIONS	2,816	843	376	421	421	932

Figure 6—Proportion of relational commands by application

The variations in use of relational commands are portrayed in Figure 6. Note that the applications A, B, C, and D include some of the menu and report generator commands summarized in the second column. The four applications listed separately account for 41.6% of all the monitored commands. In examining individual applications, the trend of high use of SUBSET (searching) continues across all applications. In most cases, PROJECTION is the next most used operation. The proportion of SUBSET to PROJECTION operations varies considerably from one application to another. The JOIN operation is generally the third most popular relational command. In the six summaries shown in Figure 6, the first three operations account for a low of 88.8% to a high of 99.9% of the basic relational commands in use. Generally, the next three operations, UNION, DIFFERENCE, and JOINALL are used rather sparingly, though enough to be an important part of many applications.

USE OF ALL COMMANDS

Command use by categories has been summarized in Figure 7. Use of prepackaged command files built either by users for their own applications or general functions provided by the implementors dominates the distribution of all commands executed. Seventy-nine percent of the commands executed are logic control commands, which can only be executed from command files. The single most used command is an IF command to do conditional testing, which constitutes over 34% of command frequency.

In prior studies of computer execution time resource use,

CATEGORIES OF COMMANDS	NUMBER OF COMMANDS	NUMBER EXECUTED	PERCENT EXECUTED
BASIC RELATIONAL	8	251 456	1.8
LOGIC CONTROL	9	11 136 528	79.0
ANALYSIS AND MODIFICATION	15	878 935	6.2
TABLE AND FILE HANDLING	13	758 046	5.4
PRINTED AND GRAPHICAL OUTPUT	11	685 039	4.9
MISCELLANEOUS	10	384 671	2.7
	66	14 094 675	100.0%

Figure 7—Summary of all commands by categories

we observed that the logic control commands make up only a small percentage of the total for an application. The basic relational commands category and the analysis and modification categories constitute the heaviest resource use. Implementors need to pay close attention to the efficiency of the commands in these two categories. Even though their frequency is low, they place a heavy load on the system.

Although some of the database input/output (I/O) operations are represented by the commands in the table and file handling category, most of the actual I/O is carried out directly and implicitly by the operating system through virtual-memory paging operations. Data are referenced directly in virtual memory. This has proven to be a fast and efficient mechanism for accessing data, since it uses an optimized path through the operating system and avoids continually having to manage record buffers and copy data to and from buffer areas. This indicates why, in the case of Regis, the frequency of I/O commands is relatively small. One normally expects a database system to have vast numbers of conventional I/O operations, although they still might be buried inside other functions with explicit subroutine calls and not appear as explicit I/O commands.

REFERENCES

1. Joyce, J. D., and N. N. Oliver, "REGIS—A Relational Information System with Graphics and Statistics," *AFIPS Proceedings of the National Computer Conference* (Vol. 45), 1976, pp. 839–844.
2. Ullman, J. D. *Principles of Database Systems.* Computer Science Press, 1980, pp. 105–109.
3. Hsiao, D. K., and M. J. Menon, "Design and Analysis of Relational Join Operations of a Database Computer (DBC)," Report OSU-CISRC-TR-80-8, The Ohio State University, Columbus, Ohio, September 1980.
4. Lacroix, M., and A. Pirotte. "Generalized Joins." *ACM SIGMOD Record,* 8 (September 1976), pp. 14–15.
5. Warn, D.R. "DO-IT: A Menu Approach to Data Management," Publication GMR-3956, General Motors Research Laboratories, Warren, Michigan, 1982.
6. Oliver, N. N., and J. D. Joyce. "Performance Monitor for a Relational Information System." *ACM 1976 Proceedings of the Annual Conference,* Houston, Texas, pp. 329–333.

Generating requirements from enterprise analysis

by DAVID V. KERNER
IBM DP Professional Center
San Francisco, California
and
ASHOK MALHOTRA
IBM Research
Yorktown Heights, New York

ABSTRACT

BICS is an enterprise analysis methodology based on a normative, top-down model of the information requirements of a corporation. This paper provides an overview of BICS and introduces REQGEN, an extension of the BICS methodology that generates definitions for the data required to manage the organization and specifications for the processes required to maintain the data.

INTRODUCTION

Most of the activity in the area of requirements for data processing applications has been in the definition of languages for specifying and analyzing requirements.[1,3,9] We feel that from an analysis of the goals and policies of an organization it should be possible to generate its data processing requirements. In this paper we discuss REQGEN a system that attempts to do this. REQGEN (requirements generator) is an extension of BICS,[5] an enterprise analysis methodology based on a normative, top-down model of the information requirements of a corporation.

The paper begins by describing BICS and a language called EAS-E[6,7] that was used to implement the BICS methodology and has heavily influenced the requirements work. It then discusses a BICS study. Finally, in the largest section, it discusses the extension of the methodology for generating requirements.

BICS

BICS (Business Information Control Study), a development of BIAIT,[2] is an enterprise analysis methodology that builds a vehicle for communicating information requirements from the end user to the data processing function. This is accomplished by building an information model of the organization that is based on the options it selects to manage the orders it receives. The methodology and the model have been successfully validated in more than 20 studies all over the world.

Orders, in BICS terms, are requests made from outside the organization that the organization must respond to. The response consists of supplying a product or a service. The product may be something manufactured (such as a CPU), a report, or even a verbal response. For each type of order the organization has certain management options, such as whether to produce to order or to stock, or whether the customer is billed or pays cash.

After the order types have been identified, a set of questions is asked about each order type to determine the management options. The answers to these questions identify the business functions. The business functions are (1) Payment: "Do you bill the customer?" (2) Delivery: "Do you keep records for use in processing the order?" (3) Customer profile: "Do you keep records, by individual customer for planning purposes?" (4) Ownership: "Do you keep title to the ordered product after delivery?" (5) Tracking: "Do you initiate service, change, or recall of a product after it has left your organization?" (6) Specification: "Do you create product/service specifications?" (7) Manufacture: "Do you make the product or service?" (8) Stock finished goods: "Do you stock the ordered product?"

A "Yes" answer to any one of the questions implies a set of data classes that must be present in the organization for the particular order type.

Next the study team takes each order type and determines which organizational units are actually accountable for the data in the data classes associated with that order type. Accountability has four aspects: i.e. accountability for (a) definition of data, (b) content of data, (c) usage of data, and (d) control over access to data.

After the relationships of organizational unit to data class are established, the study team maps organization problems, organization objectives, applications planned and installed, and databases planned and installed to the data classes.

BICS AND EAS-E

The BICS methodology described above was originally implemented manually. Apart from the sheer volume of data to be analyzed, there was also a desire to compare many organizations and suborganizations. For these and other reasons, the BICS model was implemented in EAS-E.

EAS-E is an application development system based on the entity, attribute, and set (EAS) view of system modeling. Webster's Unabridged Dictionary (second edition) defines an entity as "that which has reality and distinctness of being either in fact or for thought. . . ." In an EAS-E representation, it is usually some thing (account, check, job) of the real world to be represented in the database. It may also be a less tangible thing, like a task to be performed, which we find convenient to postulate as an entity.

The attributes of an entity can be considered to be its properties or characteristics. At any instant in time an attribute has at most one value; it may also be undefined.

In EAS-E, a set is an ordered collection of zero, one, or more entities owned by some entity. To illustrate, each account owns a set of current transactions, i.e., checks and deposits that have come in since the last monthly statement. Thus, the account for John Smith owns one such set, that for Mary Jones owns another such set, etc. In general, we say that each account owns its current transaction set. Accounts may also own a set of old transactions, outstanding loans, etc.

An entity may have any number of attributes, own any number of sets, and belong to any number of sets. An entity can both own a set of a given name and belong to a set with the same name.

These general facilities allow more specialized structures, such as trees and networks, to be expressed more or less trivially within the general framework of entities, attributes, and sets.

EAS-E requires that the system to be implemented be modeled in terms of the entities, attributes, and sets that describe its status at a point in time. The EAS-E procedural language and nonprocedural facilities can then be used to manipulate these entities, attributes, and sets in main storage and in the database.

The BICS implementation consists of (1) programs that generate data classes from the business functions and (2) programs that capture, display, and analyze the accountability information.

A BICS STUDY

A large U.S. corporation was concerned about how well its information systems (I/S) plans were tracking the corporation's business plans, and furthermore how well I/S responded to changes in the business environment. The corporation's worldwide distribution headquarters was selected as a test. The goal was to come up with a transferable management tool that would analyze the relationship between the I/S plans and the corporation's business plans.

The distribution study differed from most other BICS studies in that the area examined was a headquarters function and not an operational area. Headquarters did not move and store products, but did the overall planning for such activities. Normally, BICS information models are related to the organizational units of the entire business. This provides the analyst with the means to analyze information flow as data passes from one organizational unit to another and to identify areas where the data are potentially out of control. In the distribution example the information models were related to the functions of distribution. At headquarters there was no people or organization involvement in the information flow, and thus there could be no identification of potential data management problems.

The internal functions identified for distribution were distribution support (this included I/S and engineering), plans and control, purchasing, domestic traffic, export, import, and warehousing. The external functions identified were government, corporate finance, product service, sales, manufacturing, suppliers/vendors, and other (this included all other corporate functions outside the boundaries of distribution, such as personnel). Product service, sales, and manufacturing were, in BICS terms, the customers of distribution: They placed orders on the distribution function.

All the orders received by distribution were either for paper work to support movement of products, in which case the report or form is the product, or for the movement of the products, which is a service. Some of the orders were Request to Export, Generate Proforma, Storage Instructions, and Movement Request. For each order, the eight order management questions were asked and an information model was built. A "Yes" answer to a question generated a set of data classes from the generic BICS model stored in an EAS-E database.

The relationship between the information models and the internal and external distribution functions was now established by determining who was accountable for each aspect of each data class.

Using EAS-E, we were able to deepen the information analysis by modeling individual distribution functions. For example, we expanded warehousing into four subfunctions and built, through order analysis, information models to show information flow within the warehousing function. This was done for several other distribution functions as well.

To determine whether the BICS models could be used as a bridge between the business and I/S, we took a strategic statement from the corporation's business plans and related it to the information models. Since we had earlier built relationships between the information models and the distribution functions, we were able to determine which functions were affected by the strategic statement. If the area of the business studied had installed or planned applications, we were able to analyze the impact of the business change on existing and planned software.

The results of the study were positive. The management of the distribution function is carrying on the work, and we are now expanding BICS to enable us to generate more complete information processing requirements. This expansion is discussed in the following section.

GENERATING INFORMATION PROCESSING REQUIREMENTS

To generate requirements, the process begins, as before, by identifying the types of orders received by the organization. The eight order-handling questions are asked for each order. The answers to these questions identify the business functions. The business functions, in turn, determine the data they require and the information processes required to set up and maintain these data. These are based on the BICS data classes that have been validated in several studies.

REQGEN generates requirements on the assumption that the data are stored as entities in a central database. In other words, it generates definitions for entities, attributes, and sets that must be maintained; and for each process it specifies the operations that it performs on the EAS structures. These may be considered conceptual requirements—i.e., the data definitions can be translated into alternate-access method or database representations, and the processes can be implemented as interactive or batch programs in a conventional programming language. Alternately, the specifications can be considered EAS-E specifications: The data can be stored in an EAS-E database, and the processes can be implemented as EAS-E programs that operate directly on EAS structures.

REQGEN generates the entire data processing requirements of the organization or suborganization for which the analysis is made. If the organization has no data processing facilities, all the software can be created expediently from the REQGEN specifications. In most cases, however, the organization will already have some software in place. The REQGEN specifications will, therefore, be used to define changes that must be made incrementally to existing software.

The Data Definitions

As mentioned above, the data required depends on the business functions. For example, every organization must maintain entities that represent EMPLOYEEs and ACCOUNTs. Organizations that select the tracking option—i.e., initiate service, change, or recall of the product after it leaves the organization—must also maintain entities that represent DELIVERED PRODUCTs.

The data definitions specify attributes of each entity type and the mode (integer, text, etc.) of each attribute. They also specify the sets that the entity type owns and the sets it belongs to. The generic definition for entity type VENDOR is shown below:

ATTRIBUTE	MODE	OWNS	BELONGS	COMMENT
NAME	Text			
ADDRESS	Text			
RATING	Class			
		OFFERINGS		Product, price, qty
		PLANNED ORDERS		
		OUTSTANDING ORDERS		
		PAYMENTS OUTSTANDING		
			VENDORS	Owned by product

The mode *class* identifies a class of attributes. Members of this class are specified elsewhere.

The organization may use these definitions as they are provided by REQGEN, but it will usually want to modify them to some extent. Let us consider an example. REQGEN recommends that the entities of type EMPLOYEE have a class of attributes called personal attributes, whose function is to store personal information about the employee. This class includes the attributes NAME, ADDRESS, HOME TELEPHONE, and EMERGENCY CONTACT. A particular organization may choose to add NUMBER OF CHILDREN to this class. Or it may choose to delete some of the attributes recommended by REQGEN.

The organization also has the prerogative of changing names of entity types, attributes, and sets. This seemingly minor facility is, in fact, of great importance. The entity, attribute, and sets generate a representation of the organization. The entities in the database represent real entities in the organization. Thus, it is beneficial if data can be referred to by familiar names—e.g., WORKER instead of EMPLOYEE or COMPENSATION instead of SALARY.

There are still other degrees of freedom in the data definition phase. The organization can choose, for reasons of convenience or security, to maintain some of its information manually. The information processes that maintain this information also become manual. A computer-based process that uses this information would need to have it entered, perhaps from a terminal.

Two other decisions must be made at this time: The organization must decide whether it will run essentially separate systems for each order type, a combined system for all order types, or some combination. Organizations that have several plants must decide whether to segregate the data geographically. If the various plants handle different kinds of orders, these two decisions become interlinked. After the organization decides how the data must be aggregated, REQGEN prepares data definition files for each separate system.

To recapitulate: On the basis of the business functions for each order, REQGEN recommends the data to be maintained. The organization modifies the data according to its business needs and policies and decides how the data should be aggregated. The final data definition files can now be produced. After this is done, the data definitions can be translated into records and fields, relations, or entities and relationships, as desired; or they may be left as entities, attributes, and sets.

The Processes

There are three basic kinds of processes: processes that set up the data structures, processes that maintain the data structures, and processes that report the current state of the database.

REQGEN provides specifications for the setup processes and the maintenance processes. It does not provide specifications for the report processes. Report processes read data from the database and present it, suitably formatted for management action. Their timing, content, and, especially, format are dependent on the individual manager and are extremely variable. A query language can be used to generate many of these reports. If a program does have to be written, report programs are often the simplest programs to write.

The few processes that set up the database are used initially, for example, to set up the various ACCOUNTs. Subsequently they are used infrequently—for example, as when a new ACCOUNT is added. Setup programs are particularly simple in structure. Typically, they create an entity, give values to a few attributes, and file it into a single set. Because of their simple nature, EAS-E programs to implement these processes could be generated by a future version of REQGEN.

The majority of the process specifications generated by REQGEN are for maintenance processes. For each process REQGEN specifies generic operations to be performed on the EAS structures—i.e., the entities that must be created or destroyed or filed into or removed from sets, and the attribute values that must be updated. In most cases it cannot specify the algorithm to carry out the updating, since this will depend on the nature of the organization and its style of management.

For example, every organization that manufactures the product must have a process to produce a production plan. This process gives values to attributes that represent the production of each product by month, quarter, or other appropriate period. The production planning process must also give values to these attributes for every component of each product. Some of these components may, however, be purchased instead of being manufactured. If the component is purchased, the process must give values to attributes that specify the vendor, the quantity, etc. For each process REQGEN provides a text description similar to the one above. It also provides a list of the entity type it creates, the entity types it destroys, the attributes it reads and writes, and the sets it files and removes from.

REQGEN specifies that the production planning process is necessary if manufacturing is the type of work being done. It also specifies the entities, attributes, and sets that it works with. It does not insist that the process be carried out on the computer, and it does not specify the algorithm that must be used. A job shop may use a bill of materials explosion, whereas an oil refinery may use a linear programming approach.

Each process is triggered by an event—an external one, such as the receipt of an order or a payment, or an internal one, such as a change in product formulation or the generation of a facilities plan for the next year. Some of these events are periodic—quarterly preparation of financial statements, annual physical inventory of facilities, etc.

When an event occurs, the programs that implement the appropriate processes must be invoked. This is generally controlled manually, and manual procedures must be put in place to do so. REQGEN specifies the events that trigger each process and the organizational unit responsible for it, but it does not address the installation management procedures.

SUMMARY

Starting from an identification of the types of orders received by the organization, REQGEN generates data definitions in terms of entity, attribute, and set structures. It also generates specifications for the processes required to maintain the data and lists the events that trigger each process.

FURTHER WORK

The next step is to validate and tune REQGEN in real organizations. In a more theoretical direction, we would like to understand the causes underlying the variation in process algorithms and attempt to specify them insofar as possible. We know that one cause of variation is technology, and we feel that we can specify some of the algorithms for some of the technologies. If the algorithms can be specified, then it is not very difficult to generate the actual programs. There are precedents in this direction. Markowitz[8] describes a job shop simulation generator that captures most of the complexity (excluding accounting) of activity scheduling in a job shop. The IBM Application Customizer[4] also went a long way in generating commercial programs for specific industries. Our ultimate goal is to prepare a complete package of management software based on a description of the goals and policies of the organization. REQGEN is a first step in this direction.

REFERENCES

1. Berthaud, M. "Towards a Formal Language for Functional Specifications." In *Proceedings of IFIP Working Conference on Constructing Quality Software*. New York: North Holland, 1977, pp. 379–396.
2. Carlson, W. M. "Business Information Analysis and Integration Technique (BIAIT)—The New Horizon." *Data Base*, 10 (1979), 3–9.
3. Teichroew, D., & E. A. Hersey. "PSL/PSA: A Computer-aided Technique for Structured Documentation and Analysis of Computer-based Information Systems." *IEEE Transactions of Software Engineering*, SE-3 (1977), 41–48.
4. IBM Corporation. Hardgoods Distributers Management Accounting System. ZR30-0059-0. Available from General Systems Division, P.O. Box 2150, Atlanta, Ga. 1975.
5. Kerner, D. V. "Introduction to Business Information Control Study Methodology (BICS)." Paper presented at Symposium on the Economics of Information Processing, IBM Systems Research Institute, New York, December 15–19, 1980. Also in *The Economics of Information Processing*, vol. 1, *Management Perspectives*, New York: John Wiley, 1982, pp. 71–83.
6. Malhotra, A., H. M. Markowitz, and D. P. Pazel. "EAS-E: An Integrated Approach to Application Development." *ACM Transactions on Database Systems. Accepted for publication.* Also available as RC 8457, IBM T. J. Watson Research Center, Yorktown Heights, N.Y. 10598, 1982.
7. Malhotra, A., H. M. Markowitz, and D. P. Pazel. "The EAS-E Programming Language." RC 8935, available from IBM T. J. Watson Research Center, Yorktown Heights, N.Y. 10598, 1981.
8. Markowitz, H. M. "A Classification of Job Shop Simulation Models." RC 9301, available from IBM T. J. Watson Research Center, Yorktown Heights, N.Y. 10598, 1982.
9. Zilles, S. N., and P. G. Hebalker. "Graphical Representation and Analysis of Information Systems Design." *Data Base*, 11 (1980), 93–98.

Developing a long-range information architecture

by JAMES C. WETHERBE and GORDON B. DAVIS
University of Minnesota
Minneapolis, Minnesota

ABSTRACT

A methodology is presented for eliciting enterprise information requirements and developing a long-range information architecture. The methodology is based on a combination of business system planning, critical success factors, and ends/means analysis. The methodology is independent of organizational structure, personnel, and hardware and software; and it has been successfully implemented in a variety of organizational settings.

INTRODUCTION

Few argue that good planning is not difficult. Planning for organizationwide management information systems can be overwhelming. The Management Information Systems (MIS) Research Center at the University of Minnesota conducts an annual survey of the major corporations that sponsor it to determine the key issues that concern MIS executives. MIS planning is consistently among the top three issues.

Accordingly, the MIS Research Center conducts ongoing research in MIS planning. In earlier research conducted by the authors, a stage model of MIS planning was developed that prescribes the major steps of MIS planning and provides a framework for the more popular planning methodologies.[1,2,3] Figure 1 is an illustration of the model that categorizes well-known planning techniques in the stages of the model.

In this paper a brief overview of the planning model is provided. The second stage of the planning model, organizational information requirements analysis, is further examined; and techniques for conducting it are described. An approach to developing a long-range information architecture from the requirements is then presented.

OVERVIEW OF MIS PLANNING MODEL

The major problems of MIS planning can be defined as follows:

1. Alignment of MIS strategy with organizational strategy
2. Developing an information architecture
3. Resource allocation
4. Selecting appropriate methodologies for the previous three steps

These problems are addressed directly by the MIS planning model. The first three problems correspond to the three stages of the model. Given the framework of the model, the set of appropriate methodologies is specified for each stage; this process aids in selecting a methodology for each stage.

Practical guidance for MIS planning can be gained from the model. It can help in recognizing the nature of the MIS planning problems and in selecting the appropriate stage of planning. Too often, however, these processes are not carried out. For example, some organizations may view their MIS functions as making minimal contributions to organizational objectives. In seeking to resolve this problem, some organizations have installed a chargeout system (resource allocation planning) to make MIS pay its own way. Other organizations have conducted a business systems planning (BSP: a type of organization information requirements analysis planning) exercise to resolve the same problem. Though these activities may result in improved MIS services, the MIS planning model suggests they are probably not the appropriate methods in this situation. If the MIS effort is not responsive to the organization, the three-stage MIS planning model indicates that a strategically oriented planning effort should precede organizational information requirements planning and resource allocation planning exercises.

SELECTING A PLANNING METHODOLOGY

The three-stage planning model provides considerable insight into MIS planning issues and reduces confusion among com-

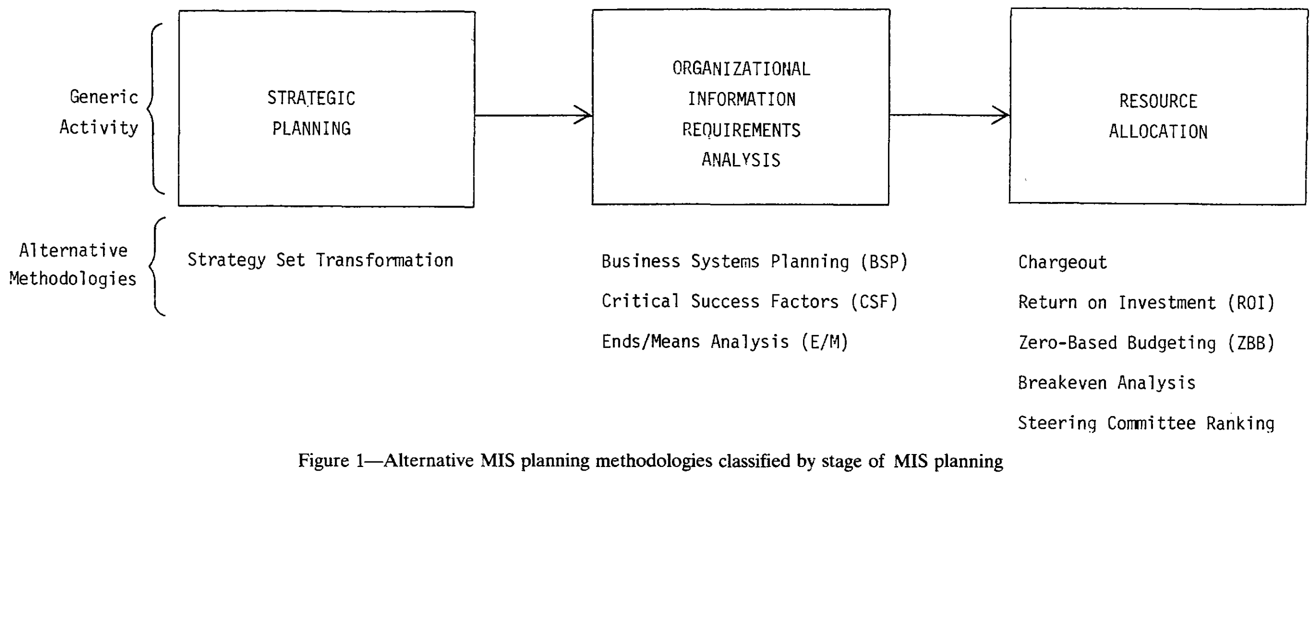

Generic Activity
STRATEGIC PLANNING
ORGANIZATIONAL INFORMATION REQUIREMENTS ANALYSIS
RESOURCE ALLOCATION
Alternative Methodologies
Strategy Set Transformation
Business Systems Planning (BSP)
Critical Success Factors (CSF)
Ends/Means Analysis (E/M)
Chargeout
Return on Investment (ROI)
Zero-Based Budgeting (ZBB)
Breakeven Analysis
Steering Committee Ranking

Figure 1—Alternative MIS planning methodologies classified by stage of MIS planning

peting planning methodologies. However, the planning model does not indicate which of several methodologies categorized in a planning stage should be used for that planning stage.

Almost no research has evaluated the comparative advantages of one technique or combination of techniques. The methodology for conducting the organizational information requirements analysis (OIRA) stage presented in this paper is based on comparative research involving three methods of enterprise requirements analysis: BSP,[5] critical success factors (CSF),[6] and ends/means analysis.[2]

CONDUCTING AN OIRA

Figure 2 portrays the model for conducting an OIRA. To make the methodology concrete, the results of a case study are used to illustrate documents generated during the study. The company agreeing to share the results of an OIRA study is EPIC Realty Services Inc., lessors of single-family dwellings. Headquartered in Washington, D.C., with offices in major cities throughout the United States, the company manages over 6,000 homes.

Define Underlying Organizational Subsystems

The first phase of the OIRA is to define underlying organizational subsystems. An organizational subsystem is a fundamental organizational activity necessary to the operation of the organization. For EPIC Realty Services Inc., the major subsystems are as follows:

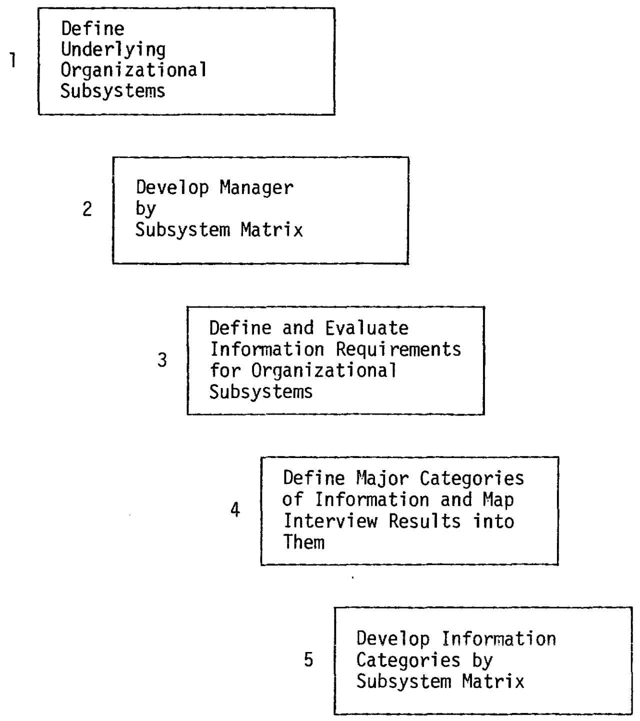

1
Define Underlying Organizational Subsystems
2
Develop Manager by Subsystem Matrix
3
Define and Evaluate Information Requirements for Organizational Subsystems
4
Define Major Categories of Information and Map Interview Results into Them
5
Develop Information Categories by Subsystem Matrix

Figure 2—Organizational information requirements planning model

1. Credit
2. Leasing
3. Maintenance
4. Evictions/delinquency
5. Marketing
6. Advertising
7. Accounts receivable/collections
8. Corporate accounting
9. Market and product analysis
10. Client reporting
11. Appraisal
12. Insurance
13. Sales
14. Personnel/administration
15. Inspections
16. Audit
17. Inventory
18. Legal

These subsystems are obtained by an iterative process of discussing all organizational activities and defining them as belonging to broad categories of subsystems. As new activities are considered, they should either be placed in categories previously defined or in a newly created category.

Develop Subsystem/Manager Matrix

Once the underlying organizational subsystems are defined, the next phase of the OIRA planning exercise is to relate specific managers to organizational subsystems. The resulting document, called a manager subsystem matrix, is illustrated in Figure 3. Note that the subsystems across the top of the matrix are the same as those identified in Phase 1.

The matrix is developed by reviewing the major decision responsibilities of each middle to top manager and relating the decision making to specific subsystems. The matrix documents the managers having major decision-making responsibility for each specific subsystem. Note that personnel changes or organizational changes can easily be reflected in an adjusted matrix.

Define and Evaluate Information Requirements for Organizational Subsystems

This phase of the planning model obtains the information requirements of each organizational subsystem by group interviews of managers having major decision-making responsibility for each subsystem. Merely asking managers to define their information requirements is frequently not satisfactory. The reasons for this are the limits of human beings as information processors.[4] Because of these limitations it is necessary to provide some structure to aid the managers in thinking about information requirements. Various methods for eliciting information requirements are basically different structures for aiding managers in the process of formulating requirements.

Research has been conducted to evaluate three approaches to structuring the set of questions for information requirements interviews. Questions based on three methods—BSP,

ORGANIZATIONAL SUBSYSTEM

	A/R	Credit	Evictions/ Delinquent	Inspection	Inventory	Marketing	Advertising	Insurance	Sale	Audit	Appraisal	Personnel/ Administration	Legal	Market & Product Analysis	Corporate Accounting	Client Reporting
Managers																
Manager 1	X	X														
Manager 2		X	X													
Manager 3						X					X			X		X
Manager 4			X	X												
•																
•																
•																
•																
•																
•																
Manager n										X		X	X		X	

Figure 3—Manager-by-subsystem matrix

CSF, and ends/means analysis—were tested. The conclusions were interesting:

1. Different managers liked different methods. No one method was dominant.
2. The methods were additive. Using more than one approach (in any order), the first method obtains the most requirements, but each additional method brings out additional requirements.
3. Since it is not possible (at this time) to know in advance the method that the manager will favor, the most efficient procedure is to use all three methods.
4. The order of use of the three techniques is in order of cognitive difficulty (the strain it puts on the managers' thought processes) and comprehensiveness.

The interview method is therefore a structured interview using questions based on BSP, CSF, and ends/means analysis. Interviews typically take 2 to 4 hours per subsystem. The maintenance subsystem at EPIC illustrates the steps of the structured interview.

Statement of purpose

The first step of the interview is to get the managers to agree on a statement of purpose for the subsystem under consideration. For example, the purpose of maintenance was defined as follows: Maintain rental property at satisfactory availability level with minimal cost and process vendor payments.

Subsystem mapping

The second step of the group interview is to define the relationship of the subsystem to all other subsystems internal to the organization or entities external to the organization. It is constructed by drawing the subsystems under consideration in the center of a chalkboard or flip chart pad and around them drawing the subsystems and entities with which they interact. Next, directional arrows are labeled and used to define the types of transactions or information flow that occurs (Figure 4).

The subsystem mapping serves as an excellent tool for making the managers aware of the full scope of the subsystem under consideration. Most interviews of this nature provide considerable enlightenment to the managers involved, since they are usually not aware of the array of activities that occur with subsystems they are familiar with.

BSP; CSF; ends/means questionnaires

After the subsystem mapping is complete, information requirements are elicited by using questions based on BSP, CSF, and ends/means analysis. The specific questions and the way they are asked are a key issue.

After interviewing several hundred managers in different organizations, we have found that the obvious question—What information do you need?—is the wrong question. It is the less obvious but properly asked indirect questions that do the job. For example, a good series of questions to ask is as

A/P MAINTENANCE

Maintain rental property to satisfactory availability level with minimal cost and to process vendor payments.

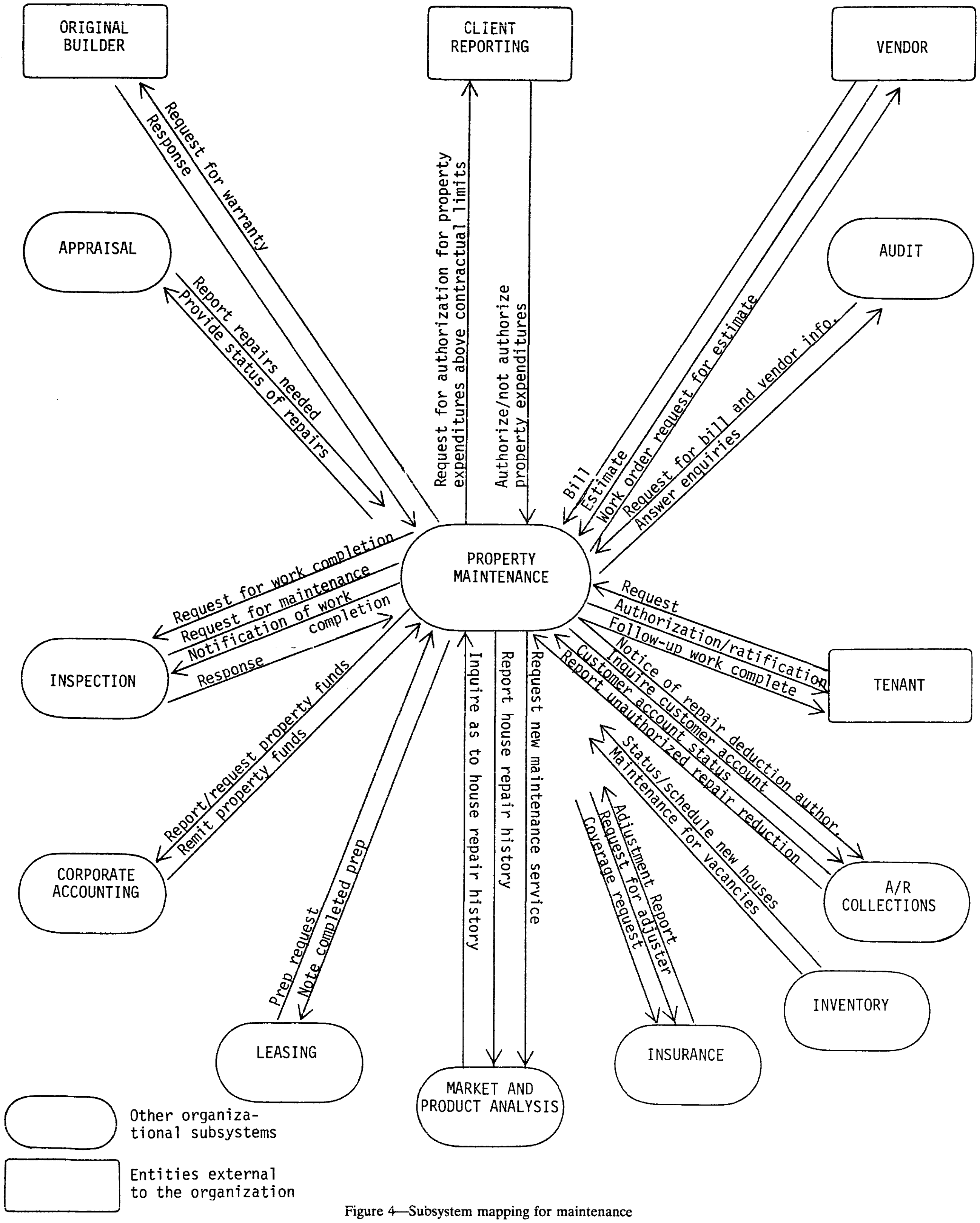

ORIGINAL BUILDER
CLIENT REPORTING
VENDOR
APPRAISAL
AUDIT
Request for warranty
Response
Report repairs needed
Provide status of repairs
Request for authorization for property expenditures above contractual limits
Authorize/not authorize property expenditures
Bill
Estimate
Work order request for estimate
Request for bill and vendor info.
Answer enquiries
PROPERTY MAINTENANCE
Request for work completion
Request for maintenance
Notification of work completion
Response
INSPECTION
Request
Authorization/ratification
Follow-up work complete
TENANT
Report/request property funds
Remit property funds
CORPORATE ACCOUNTING
Inquire as to house repair history
Report house repair history
Request new maintenance service
Notice of repair deduction author.
Inquire customer account
Customer account status
Report unauthorized repair reduction
A/R COLLECTIONS
Status/schedule new houses
Maintenance for vacancies
INVENTORY
Adjustment Report
Request for adjuster
Coverage request
INSURANCE
Prep request
Note completed prep
LEASING
MARKET AND PRODUCT ANALYSIS
Other organizational subsystems
Entities external to the organization

Figure 4—Subsystem mapping for maintenance

follows: (1) What are the major problems that this subsystem has in accomplishing its purpose? (2) How could they best be solved? (3) Can better information help? The third question reveals information requirements, but the preceding questions set the stage for the third question.

Figure 5 portrays the framework for the information requirements interview, using the three techniques: BSP, CSF, or ends/means analysis. Note that all questioning leads to the information required. The specific questions asked during the group interview are as follows:

1. Business systems planning (problems and decisions)
   a. What are the major problems encountered in accomplishing the purposes of this subsystem?
      (1) What are good solutions to those problems?
      (2) How can information play a role in any of those solutions?
   b. What are the major decisions in managing this subsystem?
      (1) What improvements in information could result in better decisions?
2. Critical success factors
   a. What are the critical success factors of this subsystem? (Most executives have four to eight of these.)
   b. What information is needed to insure that critical success factors are under control?
3. Ends/means analysis
   a. What makes goods or services provided by this subsystem effective to users?
      (1) What information is needed to insure that the subsystem is being effective at providing those goods or services?
   b. How do you define efficiency in providing goods or services in this subsystem?
      (1) What information is needed to evaluate the efficiency of this subsystem?

The interview will result in the citing of a variety of information requirements as being needed by the subsystem. A separate interview is conducted for each organizational subsystem.

Define Major Information Categories and Map Interviews into Them

The process of categorizing information is an iterative one similar to that used for defining organizational subsystems. By placing the information categories that were defined in the organizational subsystem interviews into broad, generic categories of information, an overall profile of information categories needed by the organization can be developed. Figure 6 illustrates this process.

Develop Information/Subsystem Matrix

By mapping information categories against the organizational subsystems, an information-categories-by-organizational-subsystem matrix can be developed. Figure 7 illustrates such a matrix for EPIC.

Note that at the intersections of information categories and subsystems there are coded values, defined as follows:

Utilization/Source	*Priority*
S = Source	1 = Low priority
U = Use	2 = Medium priority
B = Both	3 = High priority

Managers are asked during the interview the value of different types of information and where it might be acquired, and

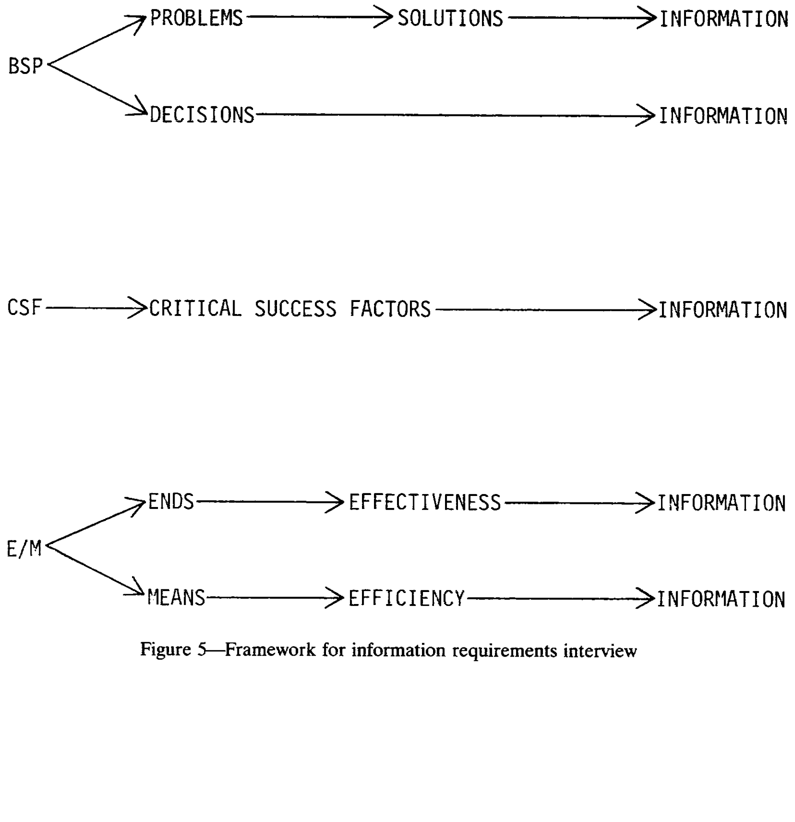


Figure 5—Framework for information requirements interview

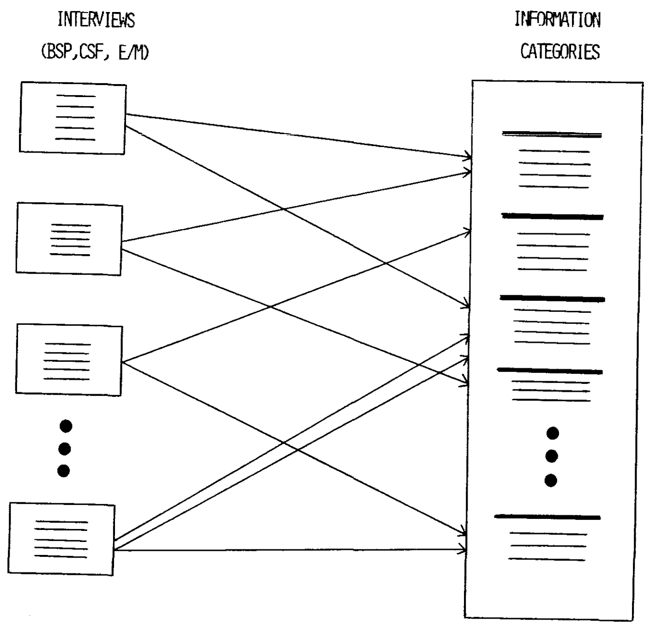


Figure 6—Interviews mapped to information categories

their responses can be coded into the table. The scores can be totaled and used as a rough indicator of the composite value of a category of information to all subsystems with which it intersects. The utilization/source also indicates whether a subsystem can generate the information needed within its own boundaries or whether it needs to obtain the information from another subsystem.

As shown in Figure 7, the source and use of information involves different subsystems. This stresses the importance of an organizationwide planning effort for information requirements analysis to avoid redundant internal generation of information among subsystems.

USE OF THE OIRA PLANNING RESULTS

The results of the OIRA exercise are twofold:

1. It identifies high payoff information categories.
2. It provides an architecture for information projects.

Identifying High Payoffs

By evaluating composite scores for information categories, the categories with the highest scores can be given first consideration for feasibility studies. Note that the information-category-by-subsystem matrix does not tell the user whether it is technically, economically, or operationally feasible to improve an information category. The matrix only indicates the relative value of the information. Feasibility studies and project definitions must still be made as usual.

Providing Architecture

By clearly defining the intersection of information and subsystems, an organization can avoid the problem of building separate, redundant information systems for different organizational subsystems. When an organization decides to improve information for one organizational subsystem, other subsystems that need such information can be taken into consideration. This avoids building separate information systems for each subsystem, which often requires reworking or duplicating what has already been done. By doing the conceptual work first an organization can identify information system projects that will do the most good and lead to cohesive, integrated systems. This is far better than randomly selecting projects that result in fragmented, piecemeal systems that are continually being reworked or abandoned because they do not mesh with the organization's overall requirements. This means planning from the top down rather than from the bottom up.

Executive's Perspective

Perhaps the best way to illustrate the value of an organization's having an organizational information architecture for MIS is to quote the president of EPIC a year after he personally led the development of their architecture:

> I had worked in top management in one of our other subsidiaries and experienced the disappointment that comes from developing systems in the traditional FIFO, piecemeal way with the consequences of redundant, non-integrated and inaccessible information.
>
> When I took over a new subsidiary, I decided there must be a better way. There was. By developing an information architecture before developing systems we have been able to pull all our systems together. Our short run system decisions are dovetailing into our long range systems. We know where we are going and getting there.
>
> Beyond that, just the process of going through an organizational information requirements analysis gave me and my management invaluable insight into our business.

REFERENCES

1. Bowman, B., G. B. Davis, and J. C. Wetherbe. "Modeling for MIS." *Datamation,* July 1980, pp. 155–162.
2. Bowman, B., G. B. Davis, and J. C. Wetherbe. "Three Stage Model of MIS Planning." *Information and Management,* forthcoming.
3. Bowman, B., G. B. Davis, and J. C. Wetherbe. "Taxonomy of MIS Planning Methodologies." *Proceedings of the Conference on Information Systems Planning,* Chicago: Society for Information Management, 1982.
4. Davis, G. B. "Strategies for Information Requirements Determination." *IBM Systems Journal,* 22 (1982), 4–30.
5. IBM Corporation. *Business Systems Planning—Information Systems Planning Guide.* Publication No. GE20-0527, Armonk, N.Y.: 1981.
6. Rockart, J. F. "Chief Executives Define Their Own Data Needs." *Harvard Business Review,* March–April 1979, pp. 81–93.

ORGANIZATIONAL SUBSYSTEMS

Information Categories	Leasing	Maintenance	A/R	Credit	Evictions/ Delinquent	Inspection	Inventory	Marketing	Advertising	Insurance	Sale	Audit	Appraisal	Personnel/ Administration	Legal	Market & Product Analysis	Corporate Accounting	Client Reporting
Contract	B/3	U/3	U/3	B/2	U/3	U/2	B/3	B/3	U/1	U/3	B/3	U/3	B/3		B/3	B/3	U/3	U/3
Policy/Training	U/3	U/3	U/2	B/3	B/3	U/3		U/3		U/3	U/3	U/3	U/3	B/3	B/2	U/2	U/2	U/2
Customer Financial	B/3		B/3	B/3	B/3		B/3		U/3		B/3	U/2				B/3		
Customer Demographics	B/3		U/2	U/2	U/2		B/3		U/3		B/3					B/3		
Complaint	S/2	B/3	B/2		U/2	B/3	B/3			U/2	B/3	U/3	S/2	B/2	U/2	B/3	B/2	B/2
Leasing/ Transactions	B/2		U/3	S/3								U/2			U/2		U/3	U/3
Vendor		B/3	B/2		B/3	B/3	B/3		B/3	B/2	B/3	U/2	B/3		B/3		B/3	
A/P		U/2	B/1		B/2				B/3		U/2	U/3	U/3	S/3	B/2		B/3	B/3
A/R		U/1	B/3	S/3	B/3		B/3			S/3	S/3	U/3			U/3		B/3	B/3
Maintenance	S/3	S/3	B/3			B/3	B/3			B/3	S/3	U/3	S/2		U/1	S/2	U/2	B/2
Warranty		B/2				B/2	B/3			B/3	U/2	U/2					U/2	
Inventory	B/3	U/2	B/2		B/3	B/3	B/3	B/3	U/2		B/3				U/1	U/3	B/3	B/2

Key:		
	S=Supply	1=low
	U=Use	2=medium
	B=Both	3=high

Figure 7—Information categories by organizational subsystems matrix

A reconfigurable VLSI architecture for a database processor

by KEMAL OFLAZER
Carnegie-Mellon University
Pittsburgh, Pennsylvania

ABSTRACT

This work brings together the processing potential offered by regularly structured VLSI processing units and the architecture of a database processor—the Relational Associative Processor (RAP). The main motivations are to integrate a RAP cell processor on a few VLSI chips and improve performance by employing procedures exploiting these VLSI chips and the system level reconfigurability of processing resources. The resulting VLSI database processor consists of parallel processing cells that can be reconfigured into a large processor to execute the hard operations of projection and semijoin efficiently. It is shown that such a configuration can provide 2 to 3 orders of magnitude of performance improvement over previous implementations of the RAP system in the execution of such operations.

INTRODUCTION

In recent years, various architectures for special-purpose processors supporting database management system (DBMS) operations directly and efficiently in hardware have been proposed.[1-10] Almost all these processors support the relational model of data[11] and provide primitives to execute the relational algebra operations on the data stored as relations. Among these database processors, the Relational Associative Processor (RAP) seems to be prominent, providing a complete set of high-level machine primitives to retrieve and manipulate data. Although the features of RAP have been described in detail elsewhere in literature,[3,7,12,13] its basic features will be summarized in the next section to aid in the upcoming discussions.

The RAP System

The RAP system hardware consists of a linear array of cells where each cell consists of a cell processor and a cell memory (CM). The tuples of the relations in the database are stored on these CMs, each of which typically holds 10^3 to 10^4 tuples, depending on tuple lengths, and all CMs are processed in parallel. The earlier prototype implementations of the RAP system have been based on head-per-track disks or CCDs; however, since high-density RAMs are becoming available (64 Kbits to 256 Kbits in the near future), it is possible to implement the CMs by using such RAM chips together with an appropriate secondary storage that will act as a virtual memory for the system.[14] The array of cell processors is controlled by an array controller, whose main functions are to broadcast search and update instructions to cells, receive the result of query programs executed over the CM contents, and interface the RAP system to a front-end system through which users of the DBMS submit queries. Figure 1 shows the structure of the RAP system. The cell processors execute the high-level instructions broadcast by the array controller on the contents of their CM usually within one scan of the memory, as the tuples in their CM are passed through the processors. Since all processors operate in parallel, the resident portion of the database can be scanned in a very short time.

The RAP logical and physical data structure is a relation consisting of tuples each augmented with several mark bit fields that act as the work space during instruction executions. These mark bits are used to denote subsets of a relation that are created as results of search operations, and the delete flag denotes deleted tuples. A typical tuple structure is shown in Figure 2. The RAP instruction set consists of simple- and cross-retrieval instructions, update instructions, and instructions for computing set functions. For more details on features of the RAP system, one can refer to various earlier papers in the literature.[3,7,12,13]

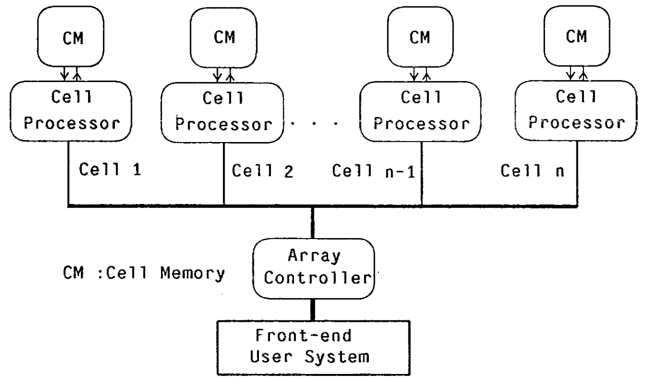


Figure 1—The structure of the RAP system

The VLSI Opportunity

In recent years the developments in microelectronics have made possible the design and implementation of VLSI computing systems based on chips that contain a large number of transistors. However, in order to use the silicon area effectively, the VLSI processing elements should be based on regular circuit structures, preferably with neighbor-to-neighbor communication. Furthermore, regular structures are simpler to design and easily extensible. Highly concurrent algorithms can be executed on such regular structures, providing very efficient solutions for computationally expensive operations.[15] The processing potential offered by VLSI is not necessarily in the speed of the individual circuits, but rather in the concurrency that can be obtained from the multiplicity of computing elements.

Previous Work on Application of VLSI to Database Processors

There has been some earlier work in applying VLSI to the domain of high-performance database processors. VLSI arrays that execute various relational operations have been proposed by Kung and Lehman.[16] These arrays are an outcome of the concept of *systolic processing.*[17] The proposed arrays have indicated a potential for high-performance relational database processing. A database processor consisting of such systolic processing subsystems can be built by connecting

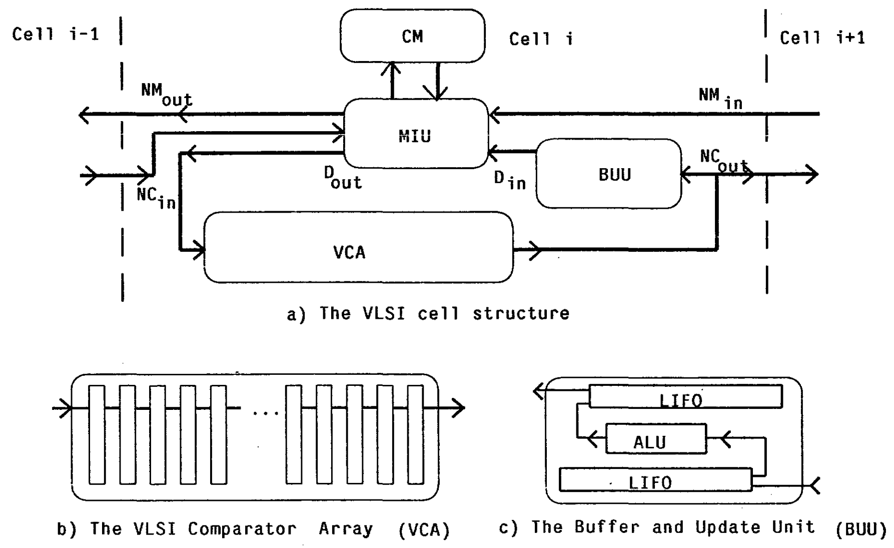

Cell i-1
CM
Cell i
Cell i+1
NMout
NMin
MIU
NCin
Dout
Din
BUU
NCout
VCA
a) The VLSI cell structure
b) The VLSI Comparator Array (VCA)
LIFO
ALU
LIFO
c) The Buffer and Update Unit (BUU)

Figure 2—The RAP data structure

the memories and processors through an interconnection network.

In conjunction with the above work, systolic processing concepts have been applied to high-speed, pipelined processing of simple relational queries and updates, which are claimed to be the more frequently executed operations on transaction-oriented database systems.[18] The proposed system is based on a multitude of systolic query arrays (SQA), each of which consists of many regularly connected processors. In this system, once a large number of simple queries are accumulated (in a very short amount of time), they are loaded into the SQAs and the appropriate relations are passed through the processors. Hence, with one pass over the relations, many simple transactions are executed.

The tree machine[19,20] is a structurally different approach to applying VLSI concepts to database processing. The proposed system consists of a set of processors organized into a tree structure and connected to a general-purpose computer. The tree machine stores and processes the database in the leaf nodes, and the internal nodes are used to route data into and out of the leaf nodes and also perform some computations. This organization can efficiently execute relational operations and sorting on data stored in the leaf nodes.

Scope and Outline of this Work

This work applies VLSI concepts to the architecture of an existing system—RAP—with the intention of (1) integrating a cell processor into a few VLSI chips, thereby decreasing cost, and (2) increasing the performance of the system in the execution of the relational operations of projection and semijoin, using system-level reconfigurability and the concurrency provided by VLSI.

After the potential bottlenecks in the RAP hardware architecture for execution of these common hard operations are discussed, a cell structure based on VLSI chips is proposed. It is then shown how dynamic reconfiguration of cell processors can dramatically increase the performance of the execution of hard operations by 2 to 3 orders of magnitude compared to earlier implementations of RAP.

The motivation for applying VLSI concepts to the RAP cell processor stems also from the fact that these cell processors are large in terms of the IC packages they use. The initial realizations of RAP cell processors[7,21] have used around 400 MSI ICs to implement a cell excluding the CM. This number has been reduced to around 160 MSI and LSI ICs in a subsequent design and implementation, resulting in a smaller and more functional system.[12,22] However, these numbers are large, if a large-scale robust system consisting of a large number of cells hosting a large DBMS is to be implemented cost-effectively. Larger physical size implies more power consumption and susceptibility to more hardware failures.

Given these considerations, it is certainly desirable to implement the cell processors with a very small number of custom VLSI chips. Concepts of regularity and modularity help reduce the complexity of designing such chips. This approach is also very cost effective: A chip, once designed and debugged, can be produced in large quantities cheaply; and system integration overheads, such as boards, cables, and cooling, are markedly reduced.

POTENTIAL BOTTLENECKS IN THE RAP ARCHITECTURE

A performance evaluation study[23] has indicated that a 100-cell RAP system can perform up to 3 orders of magnitude better than a fast uniprocessor in supporting most relational database retrieval and update operations. However, there is an important class of queries involving semijoin and projection operations where the performance of RAP is almost the same as or worse than that of a uniprocessor, despite all its parallel processing capability. The benchmarks used in two recent performance analyses of database processors[24,25] indicate this performance deficiency.

Reasons for Performance Degradation

Naive implementations of projection and semijoin operations basically involve comparing all the tuples of a relation to the tuples of another relation (or the same one, in the case of projection). On uniprocessor implementations, most such operations can be implemented by either processing the access path information or sorting the relations and then processing them. Since sorting hardware is not common in most database processors, these and other similar operations are implemented straightforwardly by iterating on one of the relations sequentially and scanning the other (or the same) relation by using the parallel processing hardware. Thus the size of the data that can be passed to the parallel scan phase in each iteration determines the number of iterations and hence performance.

The following reasons account for the performance degradation of RAP in these common hard operations:

1. Although there is parallelism at the system level (i.e., many cell processors), there is very little parallelism at the tuple processing level in every cell to help in *hard* operations.
2. Data that have to be communicated between cells during the execution of these operations are transferred over a centrally controlled bus, and there is no overlap between execution and data transfer.
3. During the execution of such operations, the cells with CM storing relations not referenced in the current instruction are idle, and their processors are not used in the execution of these operations.

A VLSI ARCHITECTURE FOR THE RAP CELL PROCESSOR

The VLSI-based RAP cell processor consists of three modules, each of which can be realized as a single VLSI chip. The CM is to be implemented by using high-density RAM chips that are currently available (64-Kbit chips) or that will be available in the near future (256-Kbit chips or larger).

The three VLSI chips that constitute the cell processor are the following (cf. Figure 3):

1. The *VLSI Comparator Array* (VCA): The VCA consists of a series of identical intelligent comparator units, each of which is made up of 8 to 16 bytes of comparand store (which can be implemented with a small RAM or a shift register), a bytewide comparator, delimiter-sensing circuitry, and a state register that keeps track of the state and configuration of the comparator. Preliminary calculations show that, depending on the technology and the comparand store size, between 20 and 80 such comparators can fit on a single VLSI chip. If necessary, similar chips can be cascaded to form comparator arrays of larger sizes. Various forms of data qualifications can be evaluated by these comparators after they are initialized by the appropriate delimiters and comparison parameters by the array controller. The high-level data qualifications of RAP instructions[3] can be evaluated on this unit. Since the emphasis of this presentation is on the execution of the hard relational operations, the details of evaluation of qualifications will not be discussed here. It is, however, necessary to state that once a tuple's values enter the comparator array (in a byte-serial fashion), they travel through all the comparators in a pipelined fashion, comparing themselves to the comparands in the comparators sensitive to their delimiters. It should also be noted that instructions that perform simple searches over tuples would need a small number of comparators (typically one to five); hence a large number of comparators in each cell would not be of much help. However, the contribution of a large number of comparators will be made evident in later sections.
2. The *Buffer and Update Unit (BUU):* This unit contains two LIFO buffers to store tuples coming out of the VCA until the result of the qualification evaluation over that tuple is known. As the tuple's closing delimiter leaves the VCA, bringing along the qualification result with itself, the tuple starts entering the ALU in a reversed fashion, and necessary mark and attribute value modifications can be made. The second LIFO reverses the tuple once again so that it goes back to the memory in the correct sequence. These LIFOs need to be large enough to hold a reasonably large tuple* and can be implemented with either RAMs or shift registers. The ALU is a bytewide unit with the standard functions. It also keeps track of various cell states during instruction execution.
3. The *Memory Interface Unit (MIU):* This unit interfaces the two processing components to the CM of the cell by performing the data transfers between them and the CM. It receives controls from the array controller. Its main functions are
   a. To read and write sections of tuples that are relevant to the current instruction being executed. Since a given RAP instruction references a small number of the attribute values in a tuple, reading and writing selective portions of tuples reduces the memory scan time considerably. We will assume that each instruction accesses the mark bit field (typically two bytes) and some additional attribute values from each tuple for comparison and if necessary for modification.

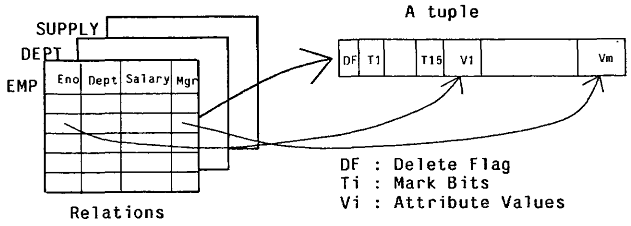


Figure 3—A cell of the reconfigurable RAP system

*Typically, 1 Kbyte to 4 Kbytes would be sufficient for most systems.

b. To insert and delete tuple and attribute delimiters to the tuple data being transferred between the CM and the two other units. All the processing circuitry in the other units is delimiter-sensitive. Hence, before data (e.g., mark bits and relevant attribute values) are passed to the processing units, they are pre- and post-fixed by tuple and attribute delimiters and accompanied by other control signals. Delimited tuples coming back to the memory are stripped off.
c. To test mark bit fields of tuples so that tuples that do not qualify for the current instruction on the basis of their mark status are accompanied by a status bit denoting that they should just pass through the processing units without being processed.
d. In addition to the above functions, during the execution of the hard operations MIU acts as a switch to reconfigure the resources. However, when all the cells are working independently in parallel, the MIU disconnects its cell from the neighboring cells, passing only its CM's contents through the processing units. This will be detailed later.

EXECUTING THE HARD OPERATIONS

This section will present structural and functional innovations that can be incorporated into a multicell RAP system based on a VLSI cell processor. It will be shown that by (1) introducing dynamic reconfigurability at the cell level and (2) employing new procedures exploiting this reconfigurability, the performance of the projection and the semijoin operations can be markedly improved, with a corresponding decrease in the execution time of the database queries using such operations frequently.

Reconfiguring the Cell Array

In this section two reconfigurations in the cell array will be presented so that the discussions in the following sections will be clearer. Let C be the number of cells in the RAP cell array. We label each cell with an integer from 1 to C, where the leftmost cell has label 1 and the rightmost cell has label C. A relation R is distributed to C_R cells having labels $r_1, \ldots, r_{C_R}$ where $r_i < r_{i+1}$ ($i = 1, \ldots, C_R - 1$).† We also define

$$REL\ (i) = Q$$

to indicate that the CM of the cell labeled i contains tuples from relation Q. We define two reconfigurations on the cell array:

1. *Processor—Reconfigure:* This reconfiguration causes all the cells in the array of C cells to connect their VCAs to form a large VLSI comparator array. The connections in the cells are established as follows: For Cell $i : 1 \leq i \leq C$, the MIU passes its NC_{in} input to the D_{out} output, hence to the input of its VCA.
2. *Memory—Reconfigure* (Q, q_i, q_j) $(q_i \leq q_j)$: This reconfiguration connects all or some of the CMs containing tuples of relation Q in order to form a large memory. The connections in the cells are established as follows:

   a. For all cells with label l, if

   $$2 \leq l \leq q_i - 1 \text{ or}$$
   $$q_j + 1 \leq l \leq C \text{ or}$$
   $$q_i \leq l \leq q_j \text{ and } REL\ (l) \neq Q,$$

   then the MIU passes NM_{in} to NM_{out} as a shift register, bypassing the cell's CM.
   b. For all cells with label l such that $q_i \leq l \leq q_j$ and $REL\ (l) = Q$, when that cell's CM is selected for processing during execution by the controller, tuples read from the CM are passed to NM_{out}, and tuples coming from NM_{in} are written to the CM. If the cell's CM is not selected for processing, then the MIU passes NM_{in} to NM_{out} as a shift register.
   c. For Cell 1, the MIU connects its NM_{out} to NC_{in} internally and hence to the input of its VCA (via the processor reconfiguration).
   d. For Cell C, the MIU connects its BUU output to NM_{in} internally so that this point becomes the write port for the large memory.

As a result of this logical reconfiguration, all or some of the cells containing tuples of relation Q are selectively enabled for processing by the large cell during the execution of hard operations.

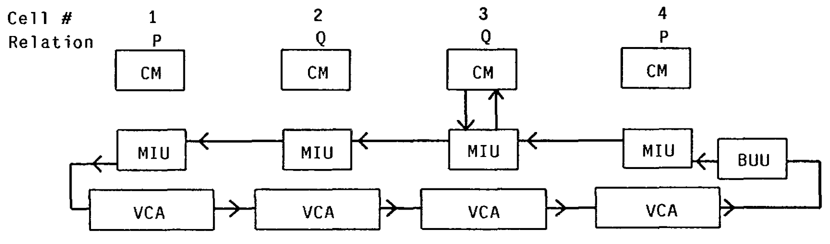
Figure 4 presents a four-cell VLSI-based RAP system in which the cell processors are connected to form a large cell. The CMs holding tuples from relation Q belong to the currently reconfigured large memory, and currently Cell 3's CM is selected for processing. The tuples in that CM are being sent to the large VCA via the MIUs of Cells 1 and 2, and the tuples returning from the processor are coming back to the CM from the BUU and MIU of Cell 4.

The Projection Operation

If a relation is viewed as a table, the projection operation removes some columns (attributes) from a relation and then removes the duplicate rows in the remaining table.[26] This operation can be implemented on uniprocessors by first sorting the columns of the relation over which the projection is to be performed and then making one pass over the sorted sequence to discard the duplicates. If access path information on the attributes is available, then such information or hashing techniques can also be used.

In the RAP architecture, this operation is programmed as an explicit iteration by using the more general GET-FIRST and RESET instructions.[3,21] At each step of the iteration, the GET-FIRST instruction gets the next unique value in the projection attribute's column, and then the RESET instruc-

†The cells over which a relation's tuples are distributed need not necessarily be consecutively labeled.



* The BUUs of the first three cells are disconnected.

Figure 4—An example of reconfiguration

tion eliminates all other tuples with the same value in that attribute by scanning all the cells in parallel and resetting their mark bits. Hence the running time of this operation is $2 \cdot v \cdot T_{scan}$, where v is the number of unique values in the projection attributes of the tuples and T_{scan} is the time to scan a cell memory completely. Before presenting the way in which projection is performed on the VLSI architecture, we introduce another primitive that is executed within the reconfigured large cell, under the control of the array controller.

Extract-Uniques $(P, a_p, M1, M2)$: The currently reconfigured large memory is passed through the reconfigured large cell, and the first $m \cdot C$ unique values from the a_p attributes of M1‡ marked tuples are loaded into the comparators where m is the number of comparators in each cell's VCA. Details are as follows:

1. Initially, all the comparators are "empty"—i.e., they have no comparands in their comparators stores, but the comparators are initialized to be triggered by the delimiter of the attribute a_p.
2. When the memory scan begins, tuples (consisting of delimited attributed values) start entering the large comparator array. The value in the a_p field is compared to the comparands of the full comparators (none if all are empty) while moving through the array in a pipelined fashion. (Note that in a given instance, an attribute value of n bytes moving through the VCA is involved in up to n different comparisons in n consecutive comparators of the large VCA and there are up to $\lfloor m \cdot C/T \rfloor$ such attribute values being processed in the array, where T is the length of relevant portions of tuples in bytes.) The value settles into the comparand store of the first empty comparator it encounters, changing the comparator state to "temporary-full," and the tuple proceeds to the BUU of the large cell without filling up any other comparator.
3. If before settling the value matches any of the values on the way, then it is a duplicate value. As that tuple's closing delimiter passes over the comparators, it senses this condition and changes the status of the temporary-full comparator back to empty. Eventually, the tuple's M1 bit is turned off in the BUU of the last cell.
4. If, however, the value does *not* match any of the values and the tuple has its M1 bit set, then as the delimiter passes over the comparators the status of the temporary-full comparator is changed to full. Eventually, the tuple's M1 bit is turned off in the BUU, and its M2 bit is set to indicate that this tuple contains a unique value.
5. If a tuple has a unique value but all the comparators are full, then the tuple's closing delimiter generates a comparator-overflow status as it exits the comparator array. The label of the cell from which this tuple came (*overflow-cell*) is returned to the array controller at the end of the memory scan. A *0* is returned if no such condition is generated. (We can also view this primitive as a function returning this value, in addition to performing the duplicate elimination.)

‡M1 is a generic name for some mark bit of the tuple; it does not specify any specific bit.

The example in Table I shows informally how the procedure described above works. In this example we assume that we have three cells in the array and a total of $m \cdot C = 3$ comparators. Initially, all the comparators are empty but sensitive to delimiter a_p. When processing starts, Tuple 1 finds the first comparator empty, and *A* settles there while the tuple's M1 mark is reset and its M2 mark is set. Meanwhile, the second tuple enters the comparator array and compares itself with the full comparator that contains ***A***, and then its value ***B*** settles into the second comparator. Following Tuple 2, Tuple 3 enters the comparator array and while passing finds that its value *A* matches the one in Comparator 1. Meanwhile the tuple goes on to settle temporarily in Comparator 3. The tuple's closing delimiter,), changes the state of this comparator back to empty. The M1 mark of this tuple is reset later in the BUU. Then the second cell is selected for processing, Tuple 4 passes over Comparators 1 and 2, and *C* settles in Comparator 3. Following it, the next tuple's value ***B*** would match the value in Comparator 2; hence its M1 mark would be reset. The next tuple would pass over all the three full comparators but would not be able to find an empty comparator; and hence none of the mark bits would be reset. Thus, the cell number 2 will be passed to the Controller, denoting the cell causing the overflow. In the remaining part of this pass, Tuple 7's M1 mark bit will be reset, and the others will pass through unchanged. The state of the CMs will be as shown in Table I.

The second pass will start with Cell 2, and the values of Tuples 6, 8, and 10 will settle in the comparators in the manner described above. Next, Tuple 10's M1 mark bit will be reset, since while passing over the comparators it matches the value in Comparator 2. There will be no overflow, and the iteration will terminate. The state of the CMs at the end of the operation is shown in Table I.

Using this and the two previously defined primitives, the

TABLE I—Example for the EXTRACT-UNIQUES operation

Initial State				After First Pass				After Second Pass				
T#	(M1	M1	a_p)	T#	(M1	M2	a_p)	T#	(M1	M2	a_p)	
1	1	0	A	1	0	1	A	1	0	1	A	
2	1	0	B	2	0	1	B	2	0	1	B	Cell 1
3	1	0	A	3	0	0	A	3	0	0	A	
4	1	0	C	4	0	1	C	4	0	1	C	
5	1	0	B	5	0	0	B	5	0	0	B	Cell 2
6	1	0	E	6	1	0	E	6	0	1	E	
7	1	0	B	7	0	0	B	7	0	0	B	
8	1	0	D	8	1	0	D	8	0	1	D	Cell 3
9	1	0	F	9	1	0	F	9	0	1	F	
10	1	0	E	10	1	0	E	10	0	1	E	

procedure for performing projection over M1-marked tuples of a relation P, over an attribute a_p, leaving tuples with the unique values marked with their M2 mark bits, is presented below.

```
PROJECT (P, a_p, M1, M2) =
  PROCESSOR-RECONFIGURE;
  overflow-cell: = $p_1$;
  repeat
        MEMORY-RECONFIGURE (P,
          overflow-cell, $P_{C_P}$);
        overflow-cell: = EXTRACT-UNIQUES
          (P, a_p, M1, M2);
  until (overflow-cell = 0);
```

After this operation is performed, the relevant columns of the relation can be read out to the front-end system.

Once the processors and the cell memories are reconfigured, the above procedure essentially passes the tuples of relation P through the comparators until the comparator array "overflows." Then for the next iteration, only the memories of the overflow-cell and those to the right of it are reconfigured to form a smaller large memory, and this memory is processed during the execution of the next *Extract-Uniques* primitive. The iterations continue until no more unprocessed unique values remain.

To compute the running time of the above procedure we first define $k_i - 1\,(\leq k_i \leq C_P)$ to be the cumulative number of CMs that can be skipped in the $(i+1)$th iteration.§ Each iteration of the above procedure takes time $c \cdot T_{scan,\,ram}$, where c is the number of memories dynamically reconfigured in that iteration and $T_{scan,\,ram}$ is the time to read the relevant section of r bytes of t byte tuples and is equal to $(r/t) \cdot T_{scan}$ (r accounts also for the delimiters). Hence, the total time the above procedure takes is

$$T_{proj,\,vlsi} = T_{scan,\,ram} \cdot (\,C_P + (\,C_P - k_1 + 1) + \ldots$$
$$(\,C_P - k_{\lceil v/(mC)\rceil - 1} + 1)),$$

since the ith:

$$\left(\left(2 \leq i \leq \left\lceil \frac{v}{m \cdot C} \right\rceil\right)\right)$$

iteration scans $k_{i-1} - 1$ less CMs than the initial iteration. The above expression can be rewritten as

$$T_{proj,\,vlsi} = T_{scan,\,ram} \cdot \left(\left\lceil \frac{v}{m \cdot C} \right\rceil \cdot C_P - \sum_{i=1}^{\lceil v/(mC)\rceil - 1} (k_i - 1)\right).$$

The Semijoin Operation

RAP uses the *semijoin* (also known as *implicit-join*) operation to answer most queries requiring the relational join operation in order to avoid generation of temporary relations. The semijoin operation is equivalent to performing the relational join[26] of the *T(arget)* and *S(ource)* relations and then projecting the resulting relation on the attributes of the relation T (the implementation does not perform the relational join, of course). Hence it selects those tuples in T that match some tuple in S. Formal treatment of the properties of this operation and a classification of relational queries that can be solved by using this operation have been analyzed earlier.[27]

In RAP this operation is implemented by using the CROSS-MARK type of instructions, which iteratively take a small number of join attribute values from the source relation and pass it to the target relation cells, marking those target tuples with a value matching any of the source values.[3,21] In order to reduce the number of such iterations, a project operation on the source relation may be performed prior to the CROSS-MARK so that duplicate source values can be discarded. However, projection itself takes considerable time.**

§More formally $k_i\,(1 \leq k_i \leq C_P)$ is the index j of cell label p_j of the cell (remember that cells of relation P have labels $p_1, \ldots, p_{C_P}$) whose CM contains the tuple having the first occurrence $(\,imC\,)$th unique value based on the ordering of the tuples of that relation with respect to their cell labels and then the obvious sequence number within the cell's CM.

**For example in RAP.2[7] projection over the source relation prior to the semi-join helps only when the ratio of the unique values to the number of tuples is less than 0.1.

In the proposed VLSI architecture, the semijoin operation is to be executed iteratively as follows: We assume that the source and target relations, S and T, occupy C_S and C_T cells respectively ($C_S+C_T \leq C$). Initially, all the cell processors are reconfigured into a single cell. In each iteration, the CMs of cells occupied by the yet unprocessed tuples of the source relation S are reconfigured into a large CM, and the next group of unique source values is loaded into the comparators of the reconfigured large cell as in the project operation. Then all the CMs of target relation cells are reconfigured into a large CM, and the target tuples are passed through the comparators. Target tuples with an attribute value that matches any of those values in the comparators are marked by the BUU of the last cell. The following primitive is required for the execution of this operation in addition to the previously presented ones.

Mark-Target (T, a_j, M3): The tuples of relation T in the currently reconfigured memory are passed through the comparator array that contains join comparand values from the source relation that are left by the preceding *Extract-Uniques* operation. Any target tuple that has a value of its a_j field matching one of the values in the comparator array is marked with the mark bit M3 in the BUU of the large cell.

The following is a more formal description of the above procedure, assuming that the source tuples to take part in the semijoin are marked with M1 bits and the resulting target tuples will be marked with the M3 bits.

```
CROSS-MARK ($T, S, a_j$, M1, M2, M3) =
    PROCESSOR-RECONFIGURE;
    overflow-cell: = $s_1$;
    repeat
        MEMORY-RECONFIGURE
                            ($S$, overflow-cell, $s_{C_S}$);
        overflow-cell: = EXTRACT-UNIQUES
                            ($S, a_j$, M1, M2);
        MEMORY-RECONFIGURE ($T, t_1, t_{C_T}$);
        MARK-TARGET ($T, a_j$, M3);
    until (overflow-cell = 0);
```

The running time of this procedure is the sum of the projection time of the source relation and the marking time of the target relation. Hence it can be written as

$$T_{join,\,vlsi} = T_{scan,\,ram} \cdot \left(\left\lceil \frac{v_s}{m \cdot C} \right\rceil \cdot (C_S + C_T) - \sum_{i=1}^{\lceil v_s/(mC) \rceil - 1} (k_i - 1) \right)$$

where v_s is the number of unique values in the join attribute of the source relation.

Performance Improvement with the VLSI Architecture

It was stated earlier that the cell processor provides only a marginal performance improvement in the execution of simple single-relation retrieval instructions. The reasons for this are that (1) such instructions require a very small amount of processing per tuple and hence do not benefit from the extra processing resources in the cell, and (2) the overall processing time depends on how fast the cell memory can be scanned, no matter how fast or powerful the cell processors are. However, the VLSI cell processors, together with reconfiguration, can provide substantial performance improvements in the execution of the projection and semijoin operations. In this section the performance improvements that can be obtained by employing VLSI and reconfigurability will be demonstrated by comparing the projection and semijoin performance of the proposed system with the earlier RAP systems' performance.

Performance Improvement in the Projection Operation

As stated earlier, RAP executes the project operation on a relation with v unique values in the projection attribute, in time††

$$T_{proj} = 2 \cdot v \cdot T_{scan}$$

To evaluate the running time of the projection operation with the VLSI architecture, we need to estimate the magnitude of the summation in the expression for $T_{proj,\,vlsi}$. For this, we present the following analysis:

Let N be the number of tuples on a single CM of the relation to be projected. Then, we can write

$$k_1 \geq \left\lceil \frac{m \cdot C}{N} \right\rceil,$$

since in the worst case all the first $m \cdot C$ tuples have unique values. In general,

$$k_i \geq \left\lceil \frac{i \cdot m \cdot C}{N} \right\rceil \geq i \cdot \left\lceil \frac{m \cdot C}{N} \right\rceil \quad i = 1, \ldots, \left\lceil \frac{v}{m \cdot C} \right\rceil - 1.$$

Hence

$$\sum_{i=1}^{\lceil v/(mC) \rceil - 1} k_i - 1 \geq \left(\left\lceil \frac{m \cdot C}{N} \right\rceil \cdot \sum_{i=1}^{\lceil v/(mC) \rceil - 1} i \right) - \left(\left\lceil \frac{v}{m \cdot C} \right\rceil - 1 \right),$$

which results in

$$\sum_{i=1}^{\lceil v/(mC) \rceil - 1} k_i - 1 \geq \left\lceil \frac{m \cdot C}{N} \right\rceil \cdot \frac{(\lceil v/(m \cdot C) \rceil - 2) \cdot (\lceil v/(m \cdot C) \rceil - 1)}{2}.$$

Therefore, the time for the projection operation can be written as

††We will assume that the CM sizes in both systems are the same, but the proposed VLSI-based system exploits the RAM by reading out relevant portions of tuples; however, this will not be used in the performance comparisons.

$$T_{proj,\ vlsi} \leq T_{scan} \cdot \left(\left\lceil \frac{\upsilon}{m \cdot C} \right\rceil \cdot C_P - \left\lceil \frac{m \cdot C}{N} \right\rceil \cdot \frac{(\lceil \upsilon/(m \cdot C) \rceil - 2 \cdot (\lceil \upsilon/(m \cdot C) \rceil - 1)}{2} \right)$$

The performance improvement (ignoring the ceiling function and the second term in the above expression) would be

$$Improvement_{proj,\ vlsi} = \frac{T_{proj}}{T_{proj,\ vlsi}} \approx \frac{2 \cdot m \cdot C \cdot t}{C_P \cdot r}$$

where t is the length of the tuples and r is the length of the mark and projection attribute values.

To convey the magnitude of this improvement, we can provide the following example: Assume a VLSI-based RAP system with the following parameters: $C = 64$, CM size = 256 KBytes, $m = 50$. The performance improvement over the earlier RAP systems in the execution of a projection operation over a relation with 128-byte tuples occupying $C_P = 8$ cells would be

$$Improvement_{proj,\ vlsi} \approx \frac{2 \cdot 50 \cdot 64}{8} = 800\text{-fold}$$

assuming that the RAMs are not read out selectively (i.e., $r = t = 128$). The improvement would be at least an order of magnitude larger if selective accessing to RAMs is also considered.

Performance Improvement in the Semijoin Operation

The time that RAP takes to execute the CROSS-MARK instruction to implement the semijoin operation is given as[23]

$$T_{join} = \left(1 + C + \frac{N_s}{n}\right) \cdot T_{scan},$$

where N_s is the total number of source tuples in each source CM, n is the number of source values that can be processed in every target relation scan, and C is the number of cells. The time the proposed VLSI approach takes was given earlier as

$$T_{join,\ vlsi} = T_{scan,\ ram} \cdot \left(\left\lceil \frac{\upsilon_s}{m \cdot C} \right\rceil \cdot (C_S + C_T) - \sum_{i=1}^{\lceil \upsilon_s/(mC) \rceil - 1} (k_i - 1) \right).$$

The improvement for a C-cell RAP system based on VLSI cells (again ignoring the second term and the ceiling function) would be

$$Improvement_{join,\ vlsi} = \frac{T_{join}}{T_{join,\ vlsi}} \approx \frac{C \cdot m \cdot (1 + C + N_x/n) \cdot t}{\upsilon_s \cdot (C_S + C_T) \cdot r}.$$

Ignoring the 1 and C compared to N_s/n, we can note that the speedup has four components:

$$Improvement_{join,\ vlsi} \approx \frac{N_s}{v_s} \cdot \frac{m}{n} \cdot \frac{C}{C_T + C_S} \cdot \frac{t}{r}.$$

The first factor indicates the speedup resulting from the characteristic of the join attribute. If the number of unique source values is very small compared to the total number of source tuples being joined, the proposed procedure will benefit. The second factor indicates the speedup due to the comparators in each cell processor. The third factor indicates the speedup due to the reconfigurability of the VLSI cell processor array. This speedup depends on the number of cells in the processor array; the maximum is $C/2$ when the source and target relations occupy 1 cell each. Finally, the fourth factor reflects the utility of reading the tuples selectively from a RAM.

To convey the magnitude of the improvement in the execution of the semijoin operation, we again give an example based on the parameters of the previous section. We further assume that $C_S = C_T = 16$ and $N_s/\upsilon_s = 10$. The improvement over the RAP.2 system[7] where $n = 5$ would be

$$Improvement_{join,\ vlsi} \approx 10 \cdot 10 \cdot 2 = 200\text{-fold},$$

again without exploiting the RAM. The improvement over the most recent implementation of the RAP.3 cell processor, designed by the author,[12,13,22] which performs semijoin with customized microcode ($n = 100$ for 16-bit numerical values) would still be at least 10-fold. However, in this case the performance improvement is mainly due to reconfiguration rather than to the large number of comparators in each cell.

Implementing Other Hard Operations

Up to this point the paper has presented performance improvements in two of the computationally expensive relational operations commonly used on the RAP system. It is also possible to implement other hard operations like union, intersection, and difference by using the same concepts and to obtain similar performance improvements, since such operations can be implemented by using the projection and semijoin operations. For example, the union operation on two relations P and Q can be implemented by treating them as a single relation and projecting them over the union attributes. The intersection of P and Q could be implemented by performing a semijoin from P to Q and then projecting the remaining marked tuples of Q over the intersection attributes. Difference of P and Q ($P - Q$) can be implemented by performing a semijoin from Q to P and then projecting the tuples of P that remain unmarked after the semijoin.

CONCLUSION

This work presents a new approach to employing the processing potential provided by VLSI to relational database processing. It is proposed that a cell processor of the RAP database processor with increased processing resources can be integrated onto three VLSI chips, each with a well-defined function. It is also demonstrated that new procedures for reconfiguring system resources during hard operations, together with the processing power provided by VLSI, can increase system performance by *2 to 3 orders of magnitude* in the execution of the hard operations compared to the current implementation. More important, the processors of a large-scale RAP system (say of 100 cells) can be implemented with a small number of processor chips (300 VLSI chips). This number is almost the same as that now required for the implementation of two cell processors of RAP.3 and less than the amount required by the earlier RAP cell processors. The savings in the auxiliary costs of boards, cables, sockets, power, and maintenance would also be considerably higher.

The operations proposed here are expected to be complementary to the RAP instructions, replacing some very common constructs that have to be implemented with more general but less efficient primitives of the RAP instruction set.

ACKNOWLEDGMENTS

This research is supported in part by the Office of Naval Research under Contracts N00014-76-C-0370,NR 044-0422 and N00014-80-C-0236,NR 048-659, in part by the National Science Foundation under Grant MCS 78-236-76, and in part by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under Contract F33615-81-K-1539.

I would like to thank Prof. Esen. A. Ozkarahan of Arizona State University, Varol Akman of Rensselaer Polytechnic Institute, and Richard Korf of Carnegie-Mellon University for their comments on an earlier draft of this paper.

REFERENCES

1. Babb, E. "Implementing a Relational Database by Means of Specialized Hardware." *ACM Transactions on Database Systems*. 4, (1979), 1–29.
2. DeWitt, D. J. "DIRECT—A Multiprocessor Organization for Supporting Relational Database Management Systems." *IEEE Transactions on Computers*. C-28, (1979), 395–406.
3. Ozkarahan, E. A., S. A. Schuster, and K. C. Smith. "RAP—An Associative Processor for Database Management."*AFIPS, Proceedings of the National Computer Conference* (Vol. 44), 1975, pp. 379–387.
4. Lin, S. C., D. C. P. Smith, and J. M. Smith. "The Design of a Rotating Associative Memory for Relational Database Applications." *ACM Transactions on Database Systems,* (1976), pp. 53–75.
5. Lipovski, G. J. "Architectural Features of CASSM—A Context Addressed Segment Sequential Memory." *Proceedings of 5th Annual Symposium on Computer Architecture,* April 1978, pp. 53–75.
6. Baum, R. I., D. K. Hsiao, and K. Kannan. "The Architecture of a Database Computer—Part I: Concepts and Capabilities." Tech. Report OSU-CISRC-TR-76-1, Computer and Information Science Research Center, Ohio State University, Columbus, Ohio, 1976.
7. Schuster, S. A., H. B. Nguyen, E. A. Ozkarahan, and K. C. Smith. "RAP-2, An Associative Processor for Databases and its Application." *IEEE Transactions on Computers,* C-28, (1979), pp.
8. Hsiao, D. K. (Ed.). "Special Issue On Database Machines." *IEEE Computer Magazine,* 12 (1979), pp.
9. Langdon, G. G. (Ed.). "Special Issue On Database Machines." *IEEE Transactions on Computers*. C-28, (1979), pp.
10. Hsiao, D. K. "Database Computers." In *Advances In Computers* (Vol. 19). New York: Academic Press, 1980.
11. Codd, E. F. "A Relational Model for Large Shared Data Banks." *Communications of ACM,* 13 (1970), pp. 377–387.
12. Oflazer, K., E. A. Ozkarahan, and K. C. Smith. "RAP.3—A Multi-microprocessor Cell Architecture for the RAP Database Machine." *Proceedings of the Second International Workshop on High-level Language Computer Architecture,* pp. 108–119.
13. Myers, G. *Advances in Computer Architecture.* New York: Wiley, 1982, pp. 429–446.
14. Schuster, S. A., E. A. Ozkarahan, and K. C. Smith. "A Virtual Memory System for a Relational Associative Processor. *AFIPS, Proceedings of the National Computer Conference* (Vol. 44), 1975, pp. 291–296.
15. Kung, H. T. "Why Systolic Architectures?" *IEEE Computer Magazine,* January 1982, pp. 37–46.
16. Kung, H. T., and P. L. Lehman. "Systolic (VLSI) Arrays for Relational Database Operations." *Proceedings of ACM-SIGMOD 1980 International Conference on Management of Data.* New York: ACM, 1980, pp. 105–116. Also available as a Computer Science Department technical report, Carnegie-Mellon University, Pittsburgh, Pa., August 1979.
17. Kung, H. T., and C. E. Leiserson. "Systolic Arrays (for VLSI)." In I. S. Duff and G. W. Stewart (eds.), *Sparse Matrix Proceedings.* Philadelphia: Society for Industrial and Applied Mathematics, 1978, pp. 256–282. Also appears as Section 8.3 in *Introduction to VLSI Systems* by Mead and Conway.
18. Lehman, P. L. "A Systolic (VLSI) Array for Processing Simple Relational Queries." In H. T. Kung, R. F. Sproull, and G. I. Steele, Jr. (eds.), *VLSI Systems and Computations.* Pittsburgh: Computer Science Press (Computer Science Department, Carnegie-Mellon University), 1981, pp. 285–295.
19. Song, S. W. "A Highly Concurrent Tree Machine for Database Applications." *Proceedings of the 1980 International Conference on Parallel Processing.* IEEE Computer Society, 1980, pp. 259–268.
20. Song, S. W. *On a High-Performance VLSI Solution to Database Problems.* Doctoral dissertation, Carnegie-Mellon University, Computer Science Department, July 1981. Also available as a Computer Science Department technical report, Carnegie-Mellon University, Pittsburgh, Pa., August 1981.
21. Ozkarahan, Esen. *An Associative Processor for Relational Databases.* Doctoral dissertation, University of Toronto, January 1976.
22. Oflazer, Kemal. "A Microprocessor Based Approach to RAP Database Machine Cell Structure—Design and Analysis." Master's thesis, Middle East Technical University, Ankara, Turkey, June 1979.
23. Ozkarahan, E. A., S. A. Schuster, and K. C. Sevcik. "Performance Evaluation of a Relational Associative Processor." *ACM Transactions on Database Systems.* (1977), pp. 175–195.
24. Hawthorn, P. B., and D. J. DeWitt. "Performance Analysis of Alternative Database Machine Architectures." *IEEE Transactions on Software Engineering,* SE-8, (1982), pp. 61–75.
25. Shultz, R. K., and R. J. Zingg. "A Performance Analysis of Database Computers." Tech. Report 82-02, Department of Computer Science, The University of Iowa, February 1982.
26. Ullman, J. D. *Principles of Database Systems.* In Computer Software Engineering Series. Pittsburgh: Computer Science Press, (Computer Science Department, Carnegie-Mellon University), 1980.
27. Bernstein, P. A., and D. W. Chiu. "Using Semi-Joins to Solve Relational Queries." *Journal of Association for Computing Machinery,* 28 (1981), pp. 25–40.

Implementing set-theoretic relational-query functions using highly parallel index-processing hardware

by SAKTI PRAMANIK
Michigan State University
East Lansing, Michigan

ABSTRACT

Hardware organizations for processing set-theoretic database query functions are presented. These organizations implement the functions by processing index trees. One advantage of this approach is that the index trees can be merged in a highly parallel fashion. Hardware organizations proposed here use the database machine approach, thus processing the index-trees on the fly. Experimental results giving the performances of these organizations are presented. Finally, a slight variation of the index tree representation, requiring much less storage for the index, is given.

INTRODUCTION

One important advantage of relational data models is that a relation is treated as a set of tuples, so a set-theoretic database query language can be designed for it.[4] These set-theoretic database query functions are a powerful tool for the database's users. But the implementation[3] of such functions on a von Neumann architecture is not efficient. This inefficiency is caused by the disk input/output (I/O) bottleneck and the lack of multiprocessing capability.[12] Several database machine architectures and architectures for very-large-scale integration (VLSI) have been proposed.[1,2,5–14] They all have multiprocessing capability. But the problem of disk I/O bottleneck for a large database still remains. This paper proposes some hardware organization suitable for implementing set-theoretic query functions. The architecture proposed here is based on processing tree type index structure. This processing is done on the fly as the indexes are being read off the secondary storage.

STRUCTURE OF INDEX TREES AND THEIR USE IN IMPLEMENTING SET-THEORETIC QUERY FUNCTIONS

One of the many ways of representing an index for a file (a set of entities) is by storing the keys in a tree structure. An example of such an index tree is given in Figure 1. Here every character of the key is represented by a node of the tree. There are two distinct advantages of structuring an index in this fashion. First, this is a more compressed representation of an index than storing each key separately would produce. Second, and more important, this tree structure allows a convenient and fast way of merging two or more indexes; thus it provides an efficient way of implementing set-theoretic database query functions, such as intersection, difference, and union. For example, the intersection function can be implemented by traveling through the index trees of both relations, and comparing their nodes. This is shown in Figure 2. Here we compare the root nodes of both trees, and find that only the character A matches. In the next level, only the children of node A need to be compared. The interesting fact is that we can discard all the subtrees under the nodes that do not match. Further, the algorithm to merge two or more index trees, can be highly parallel. The merger is sequential along the depth of the tree and parallel along its width.

This paper presents processor and memory architecture that exploits this inherent parallelism of merging two or more index trees. We also give a slight variation of this tree index that reduces the storage requirement for the trees considerably.

DIFFICULTIES OF MERGING INDEX-TREES IN PARALLEL

One difficulty of merging these index-trees is that a tree itself can be very big, big enough that only a part of it can be fetched into main memory at a time. Another is that it is difficult to exploit the inherent parallelism mentioned above, because the many subtrees that are to be merged in parallel need to be determined first, dynamically, and then loaded into the processors in parallel. We will give a database machine approach to solve this problem, where the subtrees will be determined first and then merged on the fly, as they are being read off the disk. Before we discuss this approach in detail, we would like to mention that a hierarchical sequential-storage structure will be used for storing the subtrees of an index. An example of hierarchical sequential storage is given in Figure 3. Here a tree is represented by a linear list of nodes of the index

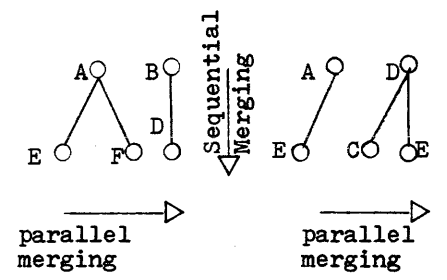


Figure 2—Direction of sequential and parallel merging on index trees

Keys
A B
A C A
A C B
D A

Corresponding Index-Tree

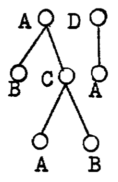


Figure 1—Index tree

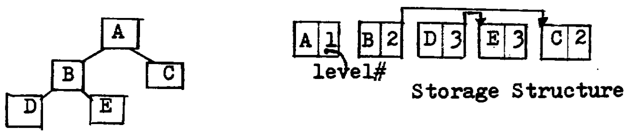


Figure 3—Hierarchical sequential storage structure

tree. Thus we can merge two index-trees by scanning the corresponding two lists, sequentially. The advantage of this storage structure is that the subtrees are readily available, right next to their parent node, in the same memory block. Also, the twin pointers help in skipping portions of the list, which accelerates the merger process.

SEQUENTIAL MERGING

The number of nodes of an index-tree of the names of 50,000 randomly chosen persons has been computed; it was found to be about 1,000,000. Though this is only an estimate, unloading data of this magnitude requires a lot of disk accesses. We will process data directly on the disk, as they are being read from it. First, we will consider sequential processing, requiring only one data stream per relation. Thus, to merge two relations we will have two data streams, each representing the nodes of the index-tree in a hierarchical sequential fashion. Each node contains the character and its level number in the tree. The level numbers are coded by a single bit as follows. A 0 indicates that the level number is one higher than the preceding node's. A bit 1 indicates that the number following is the level number of the node. An illustration of this coding scheme is given in Figure 4. The advantage of this coding scheme is that the nodes are stored in a hierarchical sequential order; thus the level numbers increase consecutively in the order in which the nodes are stored.

We use the algorithm of Figure 5 to merge two such streams. The twin pointers mentioned before are not needed, because the scanning will be sequential in a stream. In the figure, level 1 and level 2 represent the levels of the current nodes being compared in stream 1 and stream 2, and SCAN 1 and SCAN 2 represent the functions to get the next node of stream 1 and stream 2. Ch1 and Ch2 are the characters being compared.

One problem of merging the data streams at the rate of data flow is that we need to freeze a node of one stream when we compare it against a corresponding node in the other stream that has not arrived yet. Since both trees are ordered, all the intervening nodes in the second stream are discarded. We can freeze a node by using some buffer memory. The size of this buffer memory, in the worst case, must be as big as the largest index-tree; though the probability of this worst case happening is very low. We can reduce this buffer to any arbitrary size by stopping the data flow of an incoming stream when the buffer overflows. The data flow is allowed to start again when the buffer becomes available. For a disk type storage device this means a loss of a few revolutions. There is a tradeoff

```
While (not end of stream 1 or stream 2) do
IF (level 1 = level 2) then do
  IF (Ch1 = Ch2) then do
    IF (end of a key on) then do  output record pointers
        both streams
                                          Endo
    SCAN 1
    SCAN 2
     Endo
    ELSEIF (Ch1< Ch2) Then SCAN 1
    ELSE SCAN 2
    End IF
  Endo
  ELSEIF (level 1> level 2) Then SCAN 1
  ELSE SCAN 2
  Endo
  End While
```

Figure 5—Algorithm to merge two hierarchical sequential index trees

between the buffer size and the number of revolutions lost. We could, however, reduce this time loss by clustering the children nodes as shown in Figure 6. The advantage of this clustering is that the clustered nodes are the twin nodes, and all of them are available for comparison without any loss of revolution. Of course, now we will need to freeze a cluster but this will require less buffering because some tracks may now be skipped altogether.

PARALLEL MERGING

Instead of merging the index-trees in a hierarchical sequential fashion, we could also merge them level by level. Thus, we compare the characters of level 1 of both index-trees, first. Next we compare the children of these matched nodes, and so on. One serious problem of implementing this algorithm is to select from storage only the children of these matched nodes, and load them into the appropriate processors. Parallel loading is important, because the amount of data to be loaded can be very large.

Here again we will use a database machine approach to merge the index-trees. We will do this by storing an index-tree

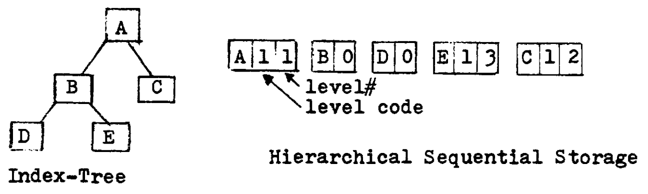
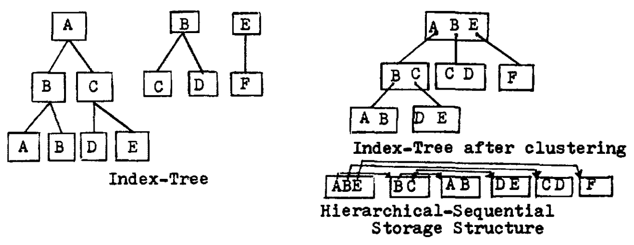


Figure 4—Coded level numbers for hierarchical sequential storage



A
B
C
A
B
D
E
B
C
D
E
F
Index-Tree
A B E
B C
C D
F
A B
D E
Index-Tree after clustering
ABE
BC
AB
DE
CD
F
Hierarchical-Sequential Storage Structure

Figure 6—Clustered index and its storage structure

in secondary memory, in two parts, and processing each part separately. The first part consists of the first K levels of the index-tree, starting with the root level. The second part contains the rest of the nodes. This is shown in Figure 7. Thus we will have as many index-subtrees in the second part as there are leaf nodes in the first part. We will store these index subtrees on the disk tracks for processing in parallel, on the fly. The extent of this parallel operation depends on the number of index subtrees there are in the second part. On the other hand, if we can read N tracks from the storage, simultaneously, there can be N parallel data flows, giving us a maximum of N parallel processings.

In merging two index trees we will first compare the first part of both the indexes. The size of the first part will be rather small and can be merged by fetching the page containing it. Typically the first part will consist of the first three or four levels, containing only about 500 to 2,000 nodes. (For a 26-character alphabet, the maximum number of nodes ever possible will be 26 for level 1 and 676 for level 2.) By merging these nodes of the first part we are able to determine which index subtrees in the second part are to be merged. We then load the appropriate instructions into the track processors for merging these index-subtrees. We assume a processor per track architecture for merging the second part. Each index subtree will be processed sequentially; they are stored on the tracks in hierarchical sequential fashion.

There are two problems with this processor-per-track architecture. The first problem is that the two index subtrees to be merged may reside on two different tracks. We need datapaths to transfer index subtrees of one track into the processor of another. Second, the two subtrees to be merged may not arrive at a processor at the same time. We solve this problem by using some buffer memory in each processor; it will be shown that the buffer size is reduced considerably by ordering the index-subtrees on the tracks, and by a proper track allocation strategy, that is, how subtrees are stored across the tracks.

DATAPATH BETWEEN ADJACENT PROCESSORS AND DISTRIBUTED TRACK ALLOCATION

Two different datapath organizations, each suitable for a particular type of track allocation scheme, are presented. The first organization provides a datapath between each adjacent

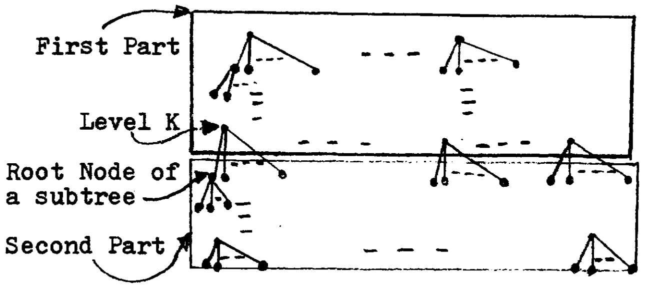


Figure 7—Partitioning index tree into two parts and creating subtrees for parallel processing

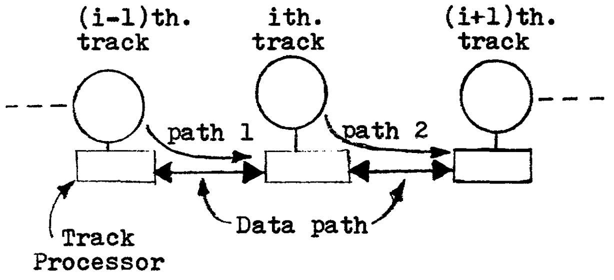


Figure 8—Hardware organization using data path between adjacent processors

processor. This is shown in Figure 8. When an index subtree is read from a track, it is transferred to the appropriate processor through this path. To avoid any path overlap of two simultaneous transfers, we schedule the merging of index subtrees. For example, there is an overlap of path 1 and path 2 in Figure 8, because the path through processor i is shared by both. We avoid this overlap by scheduling the merger, using path 1 first and then path 2 in the second revolutaion. We are able to schedule them because the track position of each of the index subtrees is known. This serialization process increases the number of disk revolutions required. But this again can be minimized considerably by ordering the index subtrees on the tracks, as well as by using a distributed track allocation scheme. In distributed allocation, all the subtrees of an index-tree are spread across as many tracks as possible. Ideally, in distributed allocation, the first K subtrees will be on the first track, the next K subtrees on the second track, and so on, where K is the ratio of the number of subtrees to the number of tracks available. Table I shows the dependence of the selectivity ratio* and the value of K on the data path congestion. It is seen that the data path congestion increases with

TABLE I—The number of disk revolutions required for selected values of K and the selectivity ratio (400 tracks)

Selectivity Ratio	$K = 1$*	$K = 2$	$k = 4$
.025	2	5	7
.125	7	9	13
.250	10	17	20
.500	12	13	14
.750	10	12	13
.880	4	6	8

*For $K < 1$, the number of disk revolutions could be further reduced.

selectivity ratio initially, but starts decreasing again with further increase in selectivity ratio. The reason for this decrease is that the higher selectivity ratio means more likelihood of the matching subtrees being resident on adjacent tracks or on the same track. In this experiment we avoided data path congestion by scheduling only one subtree per track per revolution and having no two tracks with overlapping paths in the

*The selectivity ratio is the ratio of the number of joined tuples to the number of tuples in the joining relations.

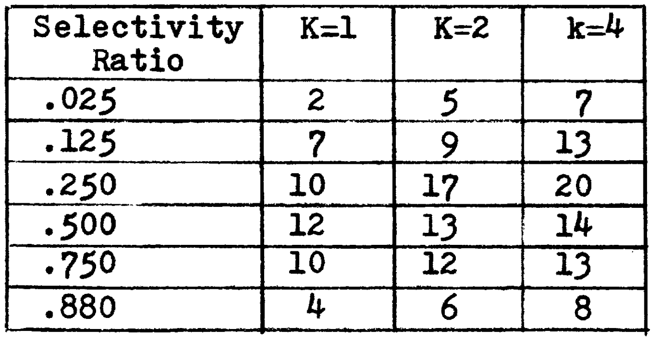
same revolution. We could, however, schedule more than one subtree on a track in the same revolution if we knew the relative rotational positions of the subtrees. This would have reduced the number of revolutions required even further.

The worst-case buffer requirement on each track depends on the numer of subtrees on a track that belong to the same index-tree. By distributing them over many tracks, we are able to reduce the size of the worst-case buffer requirement. Further, this distribution also helps in reducing the number of subtrees competing for a particular segment of the path. This helps in reducing the data path congestion also.

TRANSFER PATH BY COMMON BUFFER POOL AND CLUSTERED TRACK ALLOCATION

Though this distributed track allocation helps in reducing the worst-case buffer size as well as the data path congestion, distributing the subtrees uniformly over all the tracks may be a problem. This uniform distribution is important because it reduces the amount of buffer required per track. This is discussed in the next section. Second, path congestion may still be a problem for large numbers of subtrees.

On the other hand, transfer path by common buffer pool helps in reducing this congestion problem by clustering the subtrees within a few tracks. In this allocation scheme we store the subtrees of an index tree on the same track, and if it overflows, we allocate another track, and so on. Thus we will have as few tracks as possible for storing the subtrees of a single index tree. We use a different mechanism for data transfer between tracks here, because at any one revolution only these few tracks will be passing data back and forth, as opposed to all the tracks in the distributed track allocation. The hardware organization for data transfer path is given in Figure 9. Here we use a common transfer path between all the processors and schedule this common path sequentially among the processors by a circular priority line. Each processor keeps control of the path for no more than a preset maximum time period, and passes control to the next processor as soon as it is done with its transfer. When a processor gets control of the path, it can read from or write into a random-access buffer memory through this path. The idea here is to collect the subtrees and save them in the buffer pool until the corresponding subtrees have arrived in a processor, at which point we can start merging them in this processor. One advantage of this scheme is that we can use a common buffer pool instead of having separate buffer pools, statically, before the actual processing on the disk starts. We load each processor with the addresses in the common pool where these subtrees will be stored. Of course, these addresses will be overlapping because the memory for the subtrees that have been completely processed in the early part of the disk revolution can be reused in the later part.



Selectivity Ratio	K=1	K=2	k=4
.025	2	5	7
.125	7	9	13
.250	10	17	20
.500	12	13	14
.750	10	12	13
.880	4	6	8

Note: The value of each entry in the table represents the number of disk revolutions. For $K = 1$, this value could be further reduced. (The number of tracks was assumed to be 400 in the experiment.)

Figure 9—Data transfer by a transfer bus

A TRADEOFF BETWEEN THE SPEED OF THIS TRANSFER PATH AND THE NUMBER OF REVOLUTIONS REQUIRED

One serious problem of using such a path is the high transfer rate it requires. For example, if there are N tracks containing the subtrees of the two index trees to be merged, we will in the worst case need a path that is N times faster than the transfer rate of each of these tracks. Though the probability of this situation occurring is rather low, we can use a relatively slow transfer path by processing only a few tracks at a time. This will need a few extra revolutions to complete the merger of all the subtrees. Another advantage of processing a few tracks at a time is that the size of the buffer pool required will be even smaller.

EXPERIMENTAL RESULTS TO COMPARE THE PERFORMANCES OF THESE TWO APPROACHES OF DATA TRANSFERS

Experiments were performed to measure and compare the performances of these two organizations. We simulated the two systems with the following assumptions: only one of the two subtrees to be merged needs to be transferred to another processor; each subtree is assumed to be of the same size. These are rather simplifying assumptions, and the actual size of these subtrees depends on such factors as the type of database and the number of levels in the subtrees. We also assume that the pointers to the tuples are stored following each key in the index tree.

Table II shows the amount of common buffer pool needed

TABLE II—Buffer tree size (in number of subtrees) for selected values of the selectivity ratio and number of subtrees per track

Selectivity ratio	Subtree per track					
	240	120	60	30	20	15
.01	8	9	10	10	10	7
.10	42	59	75	81	80	84
.30	84	144	219	206	197	212
.50	95	169	282	350	326	315
.70	75	136	243	433	484	442
.90	66	99	181	318	393	410

for different selectivity ratios. First of all, it is interesting to see from column 2 that only 9.5% of the subtrees storage of an index is needed for the buffer pool. This size is very small compared to the total index database size. It is also interesting to note that the buffer pool size increases with decreasing number of subtrees per track. This is because the number of tracks required to store the subtrees increases but the buffer required per track does not decrease in proportion. We also see from this table that the buffer-pool size increases with increasing selectivity ratio at first, and then starts decreasing again with a further increase in selectivity ratio. The reason for this decrease is that the higher selectivity ratio means more likelihood of two matching subtrees being in the same relative positions on the tracks.

In distributed track allocation the buffer size per track is equal to the largest possible storage needed on a track for all subtrees of a single index. Thus, if we can distribute the subtrees of an index uniformly over the tracks, it will result in a reasonably sized buffer. Wide variations in subtree sizes, however, make this uniform distribution difficult. We can force these subtree sizes to be more uniformly distributed by making the subtrees start at a higher level in the index tree. This will create more subtrees, each one being smaller in size.

The distributed track approach will require more buffer than the clustered approach because the former allocates buffer statically among the tracks. The storage requirement as well as the data path congestion can be reduced considerably if extra revolutions are allowed. This was shown in Figure 9. Extra revolutions also help the clustered approach in reducing the buffer size and the memory speed requirement.

BINARY INDEX TREE

A slight variation of the index tree approach, mentioned earlier, will now be described. This approach requires significantly less storage, and the underlying data structure is still a tree. Further, it can be a basis for parallel merging along the width, as well as along the depth of the tree. This new tree will be called the binary index tree. We will first describe the mechanics of this binary-index-tree, and then show by experimental results the significance of this approach in saving a large amount of storage. Finally, in the Appendix, we will present a proof of this result.

It has been seen that an index tree fans out more in the first few levels of the tree than in later levels. Consequently, the index tree will have a large width at higher levels, which will therefore have a lot of nodes. We can reduce the number of nodes by partitioning the index tree vertically, as shown in Figure 10, and forming independent trees for each partition. The tree size for each partition will be very small, but we will now need a mapping between these trees. We accomplish this mapping by adding extra nodes to these trees at a logarithmic rate. These added nodes contain pointers to the corresponding nodes in the other trees. For example, an index tree can be split vertically, as shown in Figure 10, with each partition having two levels. Thus, an index tree with M levels will produce $M/2$ smaller trees. Each one of these trees, in fact, will be an index tree for keys, as if the keys are each of length

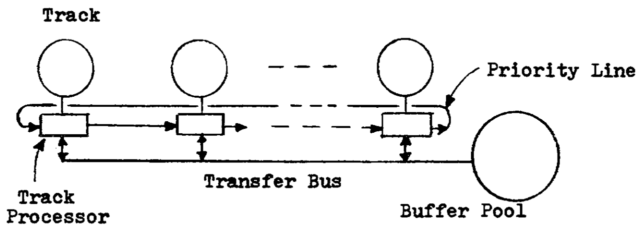


Figure 10—An example of a binary index tree

two; the first two characters of each key produce tree one, the next two characters produce tree two, and so on. Now we map a pair of these trees at a time by adding an extra level of nodes to one of them. Each of these nodes points to the leaf nodes of the other tree in the pair. Thus, we have $M/4$ trees with three levels. Next, we map these $M/4$ trees only, again a pair at a time, creating $M/8$ trees, with four levels each. We continue this process until we have one tree with $(\log_2 M) + 1$ levels. An example of this is shown in Figure 10.

A SIMPLIFIED MODEL AND EXPERIMENTAL RESULTS

We assume that the average fan-out of a node at the ith level is $Ni + 1$. The number of nodes at the root level (i.e., level 1) is $N1$, say. Then in the ith level the number of nodes will be $N1 \times N2 \times \ldots \times Ni$. Thus the total number of nodes, S, for an index tree M levels deep, is:

$$S = \sum_{i=1}^{M}\left(\prod_{j=1}^{i} N_i\right). \qquad (1)$$

The number of nodes, $S1$, for a binary index tree will be

$$S1 = N1\frac{M}{2} + \sum_{i=1}^{\log_2 M} \frac{M}{2^i}\left(\prod_{j=1}^{2^i} N_j\right). \qquad (2)$$

Using these two formulas we have computed the values of S and $S1$, for different values of the sequences $N1 \times N2 \times \ldots \times NM$. These are given in Table IV. Each group size is defined in Table III. It is seen that the amount of storage needed in binary index tree is less than in the index tree approach. Further, this saving increases with increasing length of the keys. For example, the saving is 66.4% for keys of length 16, and is 72.53% for keys of length 32. It is also seen from Table IV that the percentage gain for the binary index tree is less for faster growth of the sequences $N1, N2, \ldots, NM$.

It is important to note here that the number of nodes required in the binary index tree approach is considerably less but some of the nodes need extra storage for the mapping. It turns out, though, that the amount of this extra storage is much less than the large amount saved by reducing the number of nodes. Finally, it is shown in the appendix that for any value of the sequence $N1, N2, \ldots, NM$, the number of nodes for the binary index tree is less than or equal to the number of nodes for the index-tree.

TABLE III—Average fanout at each level of the index tree of a group

Group#	$N1$	$N2$	$N3$	$N4$	$N5$	$N6$	$N7$	$N8$	$N9$	$N10$	$N11$	$N12$	$N13$	$N14$	$N15$	$N16$–$N64$
1	26	11	7	4	2	1.4	1.2	1.1	1.01	1.1	1.05	1.1	1.05	1.2	1.045	1.
2	26	11	8	4	1.44	1.25	1.3	1.1	1.01	1.10	1.2	1.3	1.1	1.113	1.	1.
3	26	12	8	4	2.4	1.4	1.1	1.	1.15	1.10	1.	1.05	1.02	1.	1.	1.
4	26	18	10	5	1.5	1.1	1.2	1.08	1.	1.	1.	1.	1.	1.	1.	1.
5	26	18	12	8	1.113	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.

*An index with 50,000 keys is assumed.

APPENDIX

Proposition The number of nodes, S, of an index tree is greater than or equal to the number of nodes, $S1$, of a binary index tree for any values of the sequence $N1, N2, \ldots, NM$.

Proof. From (1) and (2) we have

$$S = \sum_{i=1}^{M} \prod_{j=1}^{i} N_j$$

$$S1 = N1\frac{M}{2} + \sum_{i=1}^{\log_2 M} \frac{M}{2^i}\left(\prod_{j=1}^{2^i} N_j\right).$$

In general, for a large number of keys, S and $S1$ are fairly large, and we ignore the first term from both S and $S1$. Thus we have

$$S = \sum_{i=2}^{M}\left(\prod_{j=1}^{i}(N_j)\right)$$

$$S1 = \sum_{i=1}^{\log_2 M}\left(\frac{M}{2^i}\right)\prod_{j=1}^{2^i}(N_j)$$

Note that the last terms of S and $S1$ are the same. Since we want to compare the values of S and $S1$, we can ignore this similar term from both. Thus from S we get

$$S' = \sum_{i=2}^{M-1}\left(\prod_{j=1}^{i} N_j\right)$$

and from $S1$ we get

$$S1' = \sum_{i=1}^{(\log_2 M)-1} \frac{M}{2^i}\left(\prod_{j=1}^{2^i} N_j\right).$$

Now S' can be rewritten as

$$S' = \sum_{i=1}^{(\log_2 M)-1}\left(\prod_{j=1}^{2^i} N_j\right)\left(1 + \sum_{j=1}^{2^i-1}\left(\prod_{k=2^i+1}^{2^i+j}(N_k)\right)\right)$$

TABLE IV—Percentage savings in number or nodes achieved by using binary index trees, by key size and group

	Key size		
Group	16	32	64
1	66.4	72.53	74.37
2	66.24	73.2	75.2
3	68.33	71.11	72.11
4	58.65	61.03	61.9
5	48.75	50.56	51.3

Thus the term $\prod_{j=1}^{2^i} N_j$ is same as in $S1'$. We abbreviate this term as C_i. Thus,

$$C_i = \prod_{j=1}^{2^i} N_j.$$

It should be noted that $C_i \le C_i + 1$ for all is, because $N_i \ge 1$ for all is.

Now we can write S' and $S1'$ as

$$S' = \sum_{i=1}^{(\log_2 M)-1} C_i\left(1 + \sum_{j=1}^{2^i-1}\left(\prod_{k=2^i+1}^{2^i+j}(N_k)\right)\right)$$

$$S1' = \sum_{i=1}^{(\log_2 M)-1} \frac{M}{2^i} C_i.$$

The term within parentheses for S' satisfies the following condition,

$$1 + \left(\sum_{j=1}^{2^i-1} \prod_{k=2^i+1}^{2^i+j} (N_k)\right) \ge 2^i,$$

because there are $2^i - 1$ terms in the summation, each of which is greater than or equal to one. Thus we have

$$S' \ge S'' = \sum_{i=1}^{(\log_2 M)-1} 2^i C_i. \qquad \text{(A1)}$$

By subtracting $S1'$ from S'' we get

$$S'' - S1' = \sum_{i=1}^{(\log_2 M)-1} C_i\left(2^i - \frac{M}{2^i}\right)$$

The absolute values of $2^i - M/2^i$ for $i = 1$ through $(\log_2 M) - 1$ form a duplicating sequence. For example, its values for $i = 1$ and $i = (\log_2 M) - 1$ are $2 - M/2$ and $-(2 - M/2)$, respectively. Similarly, for $i = 2$ and $i = (\log_2 M) - 2$ the values are $(4 - M/4)$ and $-(4 - M/4)$, respectively, and so on. Thus, we can write

$$S'' - S1' = \sum_{i=1}^{N}\left(\frac{M}{2^i} - 2^i\right)(C_{2N+1-i} - C_i)$$

where $N = (\log_2 M - 1)/2$.

For this range of values of i we have

$$(M/2^i - 2^i) \ge 0$$

and

$$(C_{2N+1-i} - C_i) \ge 0.$$

Thus

$$S'' - S1' \geq 0. \tag{A2}$$

From relations (A1) and (A2) we get

$$S' - S1' \geq 0,$$

which was to be demonstrated.

REFERENCES

1. Babb, E. "Implementing a Relational Database by Means of Specialized Hardware."*ACM TODS,* (1979), No. 1.
2. Banerjee, J., D.K. Hsiao, and K. Kanna. "DBC—A Database Computer for Very Large Dtabases." *IEEE Transactions on Computers,* C-28 (1979), No. 6.
3. Blasgen, M.W., and K.P. Eswaran. "Storage and Access in Relational Databases." *IBM Systems Journal,* 16 (1977), No. 4.
4. Date, C.J. *An Introduction to Database Systems* (3rd ed). Reading, Mass.: Addison-Wesley, 1981.
5. DeWitt, D.J. "Direct—A Multiprocessor Organization for Supporting Relational Database Management Systems," *IEEE Transactions on Computers,* C-28 (1979), No. 6.
6. Hawthorn, P., and D.J. DeWitt. "Performance Evaluation of Database Machine Architectures." *Proceedings of the Seventh International Conference on Very Large Date Bases,* 1981.
7. Kung, H.T., and D.L. Lehman. "Systolic (VLSI) Arrays for Relational Database Operations." *Proceedings of the ACM-SIGMOND Conference,* 1980.
8. Menon, M.J. and D.K. Hsiao. "Design and Analysis of a Relational Join Operation for VLSI." *Proceedings of the Seventh International Conference on Very Large Data Bases,* 1981.
9. Ozkarahan, E.A., S.A. Schuster, and K.C. Smith. "RAP—Associative Processor for Database Management." *AFIPS, Proceedings of the National Computer Conference* (Vol. 44), 1975.
10. Pramanik, S. "Implementing Relational Join by Database Filters," Technical Report, Computer Science Department, Michigan State University, 1982.
11. Shaw, D. "A Relational Database Machine Architecture," *Proceedings of the Fifth Annual Workshop on Computer Architecture for Non-numeric Processing,* Pacific Grove, California, 1980.
12. Song, S.W. "A Survey and Taxonomy of Database Machines," Quarterly Bulletin, IEEE Computer Society Technical Committee on Database Engineering, December 1981.
13. Su, Stanley Y.W., and G. Jack Lipovsky. "CASSM: A Cellular System for Very Large Databases," *Proceedings of the International Conference on Very Large Databases,* 1975.
14. Yao, S.B., and Fu Tong. "Design of a Two-Dimensional Join Processor Array. *Proceedings of the Sixth Annual Workshop on Computer Architecture for Non-Numerical Processing,* Hyeres, France, 1981.

Cost-effective ways of improving database computer performance

by DAVID K. HSIAO
Naval Postgraduate School
Monterey, California

ABSTRACT

In this paper the hardware features that characterize the performance bottlenecks of conventional database computers are identified. Motivations for and proposals of new architectures for overcoming the bottlenecks for future database computers are given.

BACKGROUND

According to the 1974 publication on its database computer as a backend of the mainframe host computer for database management,[1] XDMS was aimed to provide the following:

1. Cost-saving and performance gain through dedicated software and backend hardware
2. Shared databases
3. Centralized protection
4. Ease in software development on a standalone backend and for the backend

The last three aims had largely been achieved. For example, XDMS was able to communicate not only with the original host, a Univac 1108, but also with a new host, an IBM 360. In other words, XDMS allowed two host computers (more specifically, their respective application programs) to access the same database, even though the application programs running on the one host cannot be run on the other host. Obviously, XDMS was provided with multiuser and multirequest supports and update locks. By allowing only the backend to control, manage, and access the database store, namely the disks, XDMS achieved the aim of centralized protection. For the first time, the disks of the database could be physically protected from the disks of the mainframe. The only possible way of compromising the protected database was by way of the "front door," through the host-backend communications. Data security employing encryption could ease such "frontal" attack. Finally, it was obvious for the developer that software development could always be accommodated readily if the machine for which the software was aimed was present and if a new machine could be used for software development. What was not clear at the time was the success of achieving the first aim—the cost saving and performance gain through dedicated software and hardware.

Cost saving was arguable. The software development and purchase cost had been shown to be, at worst, comparable to the software development and purchase cost of the database management system (DBMS) for the mainframe. The hardware cost of the backend had not been higher than the hardware cost of mainframe upgrade for the DBMS. For example, the original backend of XDMS was a 16-bit minicomputer known as Meta-4, which cost $60K in 1974. Since the cost of minis is rapidly coming down, the same minicomputer should cost less than $30K now. In fact, it would be cheaper than a disk controller. Disks for the database were needed whether the backend was utilized or not. On the other hand, if the backend were not employed, the DBMS acutely needed the addition of main memory, processing power, and channel capacity for its support in the mainframe. Thus, the cost saving by way of the database management backend was indeed arguable.

The major disappointment was in performance gain. Performance may be measured in terms of the following: (a) the communication and transmission times between the backend and its host, and (b) the transaction execution time. XDMS was reported to have very low degradation due to communication and transmission. It was also reported to have good performance for complex queries. The latter report was difficult to justify in view of the small size (i.e., 1,500 records) of the database involved. Most database system designers knew that the time complexity of certain queries (e.g., relational joins) was proportional to the size of the database involved (e.g., cardinality of the relations involved). The time complexity of a query was a determining factor of the query execution time. Thus, the larger the database there was, the longer the transaction time would be. Perhaps the argument on performance gain was made in terms of the *relative* performance of the transaction execution time; that is, given the same set of transactions, the transaction set was first executed in a host without a backend and then executed in the backend. By comparing their response times (i.e. the communication and transmission time, plus the transaction execution time) relative to each other, we could then conclude one's gain over the other. Subsequent experiments with XDMS-like backends yielded no appreciable performance gain over the conventional mainframe-oriented ones. This lack of performance gain was due to two factors: First, it was difficult, if not impossible, to come up with software design that ran appreciably faster in the backend but not in the mainframe. Second, the backend, as a general-purpose minicomputer, was inherently slower in speed and meager in resources than the well-endowed host. Thus, database management software did not necessarily run faster on the backend.

Others attempted to introduce, instead of a dedicated mini, specially configured hardware to speed up the transaction execution time. These new backends, such as Britten-Lee's IDM 500 and Intel's iDBP are mostly microprocessor-based hardware with a generous use of random-access memories. However, even with these attempts, the performance gain may still be elusive. Let us examine the architecture of current database computers and identify their performance bottlenecks.

WHERE ARE THE PERFORMANCE BOTTLENECKS?

Consider the architecture of a backend computer depicted in Figure 1, where conventional disks are used for the database

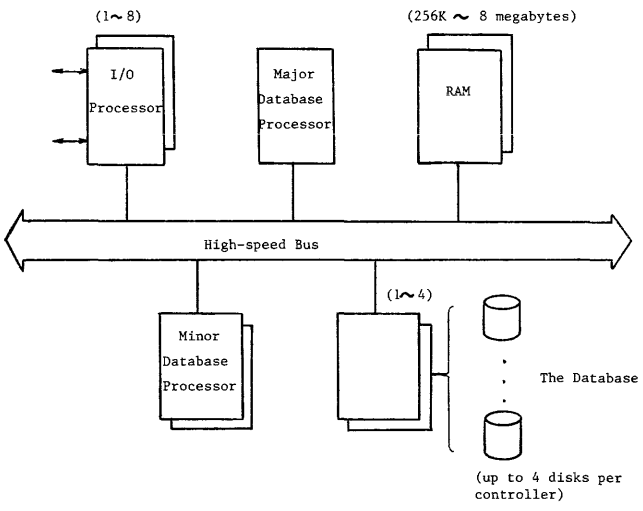


Figure 1—Typical configuration of a microprocessor-based database computer

store. It is important to note that everything is hung on the high-speed bus. In other words, for the data coming off the disk and going into the random-access memories, coming off the memories and going toward the hosts or terminals, or being accessed by the major database processor and other minor database processors, the high-speed bus is the only throughway for the data movement and access. Consequently, the performance of the backend cannot exceed the capacity of the high-speed bus. Presently, a typical high-speed bus has a transfer rate of 20 to 320 megabits/sec.

For a database computer with four disk controllers, each of which has four disk drives, assuming that the disk is of medium capacity, 300 megabytes/drive, we have the following calculations:

$$
\begin{aligned}
4 \text{ (controllers)} \times 4 \text{ (drives)} \times 300 \times 10^6 \text{ (bytes/drive)} &= 48 \times 10^8 \text{ bytes of the database} \\
&= 384 \times 10^8 \text{ bits,} \\
384 \times 10^8 / (20 \times 10^6) &= 1{,}920 \text{ seconds to read out the entire database} \\
&= 32 \text{ minutes.} \\
384 \times 10^8 / (320 \times 10^6) &= 120 \text{ seconds} = 2 \text{ minutes.}
\end{aligned}
$$

This indicates to us that for text search and retrieval, an important application of database management, it takes at least 2 to 32 minutes to search and retrieve the entire textual database. Even if the search and retrieval is restricted to a fraction—say a quarter—of the database, it will require at least one-half minute to 8 minutes. Consequently, conventional database computers are not suitable for text search and retrieval.

For formatted database management, where indices are used and accesses to the database are more selective, we do not use disk drives as the basis for calculation. Instead, we consider that the physical records correspond to disk tracks, which are the units of data access and transfer. Assume that a track is of 24 Kbytes. We then have the following:

$$(20 \times 10^6 / 8) / 10^3 = 2{,}500 \text{ byte/msec (the bus capacity),}$$

$$24\text{K} / 2.5\text{K} = 10 \text{ msec}$$

to place a track of data in the main memories. Similarly, at the higher rate of 320×10^6 bits/sec, we need .6 msec to place a track of data in the main memories. We assume the optimal situation, that only one track of data is needed from each disk drive and that all the track seeks have been overlapped. Thus, all 16 disk tracks coming from 16 separate drives can be read back-to-back at the maximal bus rate. We then need

$$16 \times 10 = 160 \text{ msec and } 16 \times .6 = 9.6 \text{ msec.}$$

Since, for formatted databases, it uses indices and auxiliary information to select data tracks, the database computer must access the indices and auxiliary information that are also stored on the disks. Thus we need another 9.6 to 160 msec. Altogether, then, we need a minimum of 9.6 to 160 msec to get a unit of data into main memories for processing. Assuming that the CPUs, with cycles ranging from 200 nsec to 1 μsec, of most of the 16-bit microprocessors and minicomputers can keep up with the incoming data by executing short system programs, we quickly see that to improve the performance we must improve the transfer rate of the high-speed bus, that is reduce the time needed to transfer a unit of data.

Although a higher speed (say, beyond 320 megabits/sec) of the bus is attainable at higher cost, the gain in bus speed is nevertheless offset by the higher capacity of the disk (say, of 1.25 gigabytes/drive, which would make a database of 20 gigabytes a reality). Larger-capacity disks imply larger databases; in turn, these imply wider distributions of related data on the disks, which in turn imply more data pages needed in the main memories. Consequently, the higher-speed bus is used for disks of ever larger capacity. With these new figures and assumptions, we may go through the same calculation again. We then discover that a backend is required to pay a fixed "cost" (of approximately 9.6 to 160 msec) for a unit of data management and data transfer, despite the large (i.e., gigabyte) capacity of the disks and the high cost of the faster bus used for the backend. In fact, the use of such an expensive bus with large disks will not show any performance gain over the use of a less-expensive bus with medium disks. By now, we learn that our earlier conclusion is not valid. The cost-effective way of improving the database computer performance does not lie in the expensive higher-speed bus, since larger-capacity disks for very large databases will offset any possible reduction of the fixed overhead in data management and transfer. The question is therefore whether or not we can improve the database computer performance, despite the presence of the fixed overhead and the lack of any prospect of taking advantage of the advancement of bus and disk technology.

ARCHITECTURAL SOLUTIONS FOR IMPROVING THE PERFORMANCE

There are two proposals. Although they are different in their architectural approaches and technological choices, these proposals use the same principle. The principle is sharing the same fixed cost (overhead) by using a multiplicity of identical hardware and by processing the data on the identical hardware concurrently.

Database Stores with Built-in Parallel Processing Logic

Consider our first proposal, which is depicted in Figure 2. In this proposal, the moving-head disks are modified so that they can perform parallel read-out and write-in operations. Such technology was reported as early as 1978.[2] The disk controller is also modified so that there is a processing unit for each data steam coming from the track. Therefore, there are as many processing units in the controller as there are tracks in a cylinder. Assuming 20 tracks per cylinder, we need only 20 processing units in a controller. Each one of the 20 processing units has an identical microprocessor-based architecture with considerable use of random-access or shift-register-like memories. Each unit is essentially a major database processor of the sort depicted in Figure 1. The differences between Figure 1 and Figure 2 are that

1. The same major database processor is multiplied 20 times in the new disk controllers.
2. Different processing units process their own data streams coming from different tracks.

The first difference has been to some extent overcome, since the manufacturer of the controllers has already incorporated processing logic for defect detecting and error decoding into the controller, as well as logic for executing the software online I/O routines (known as access methods). There is no reason why the disk manufacturers cannot make the controller even more intelligent. The second difference can be overcome since we can achieve a microprocessor-based architecture where single-instruction-and-multiple-data-stream (SIMD) and multiple-instruction-and-multiple-data-stream (MIMD) modes of database management become a reality. Thus, all the data streams "share" the same fixed overhead; meanwhile, 20 times as much data access, transfer and processing may be accomplished.

This approach to reducing overhead and improving performance is cost-effective because the disk technology for parallel read-out and write-in is here, the controller technology for built-in logic is also here, and the addition of identical hardware is proportional to the performance gains. We observe that in multiplying the hardware in the controller we are asking not for ever-faster buses, but for multiplication of the existing bus and processor structure. This proposal has been thoroughly analyzed and studied.[3]

Software Multibackends

Consider our second proposal, which is depicted in Figure 3. This proposal is aimed at addressing the following question. Is it possible to use a multiplicity of minicomputers and their disk systems, unconventional hardware configuration, and innovative software design for achieving improvements in throughput and response-time for large and growing databases over what conventional and single-backend database management can provide?

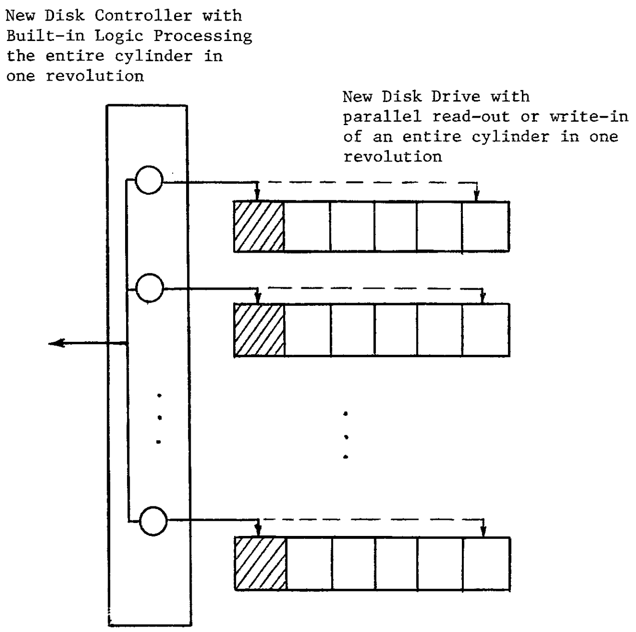

New Disk Controller with Built-in Logic Processing the entire cylinder in one revolution
New Disk Drive with parallel read-out or write-in of an entire cylinder in one revolution

Figure 2—High-performance disk system

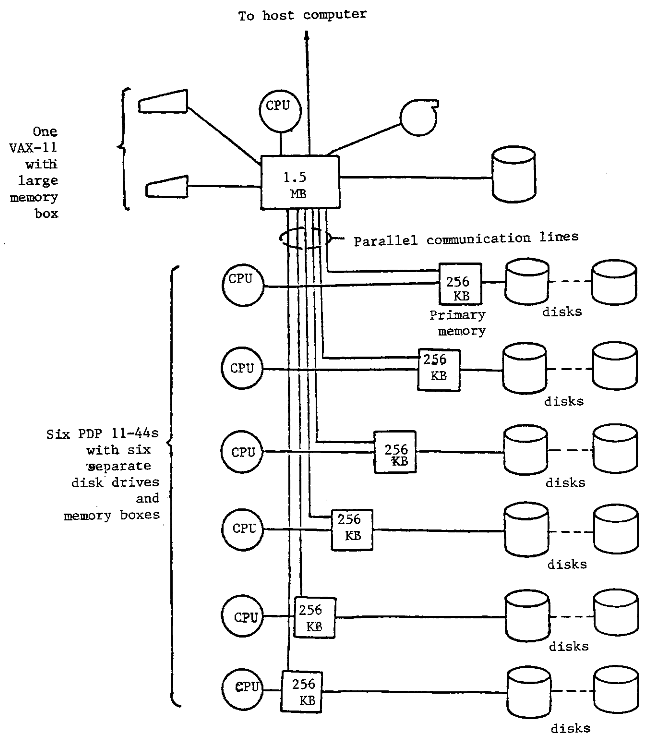

To host computer
One VAX-11 with large memory box
CPU
1.5 MB
Parallel communication lines
Six PDP 11-44s with six separate disk drives and memory boxes
CPU
256 KB
Primary memory
disks

Figure 3—Multibacker database management system

Measures of a good multibackend database system are that

1. The throughput improvement is proportional to the multiplicity of the backends.
2. The response time is inversely proportional to the multiplicity of backends.
3. The system is extensible for capacity growth and/or performance improvement.

The cost-effective ways of extending the software and hardware of the multibackend system with one controller (i.e., master) and several backends (i.e., slaves) are to

1. Allow the addition of more backends of the same type, instead of the replacement of the present backends with more powerful models
2. Require identical software in each of the backends and replicate the existing software on new backends
3. Minimize the role of the controller of the backends

We note that the addition of the same type of minis and disk systems, our first way, will incur few system interruptions. Intuitively, for the same database we double the number of minis if we want to double the performance gain. For a growing database, say, that will double the current database size, we would double the hardware to maintain the present performance. Our second way of extending the system, software replication, can be easily accomplished on the new hardware by doing a system-generation (i.e., SYSGEN); furthermore, the increase of software replication does *not* imply the increase of software complexity, since all the software replicated in the slaves is identical. Finally, the master should not become the bottleneck of the new and existing slaves, else there will be little extension of the system capacity and little improvement in system performance. To keep the master from becoming a potential bottleneck, we require that the controller (i.e., master) perform minimal (yet necessary) work. Presently, a software multibackend prototype is being implemented.[4,5] Projected performance gain (or loss) will soon be validated (or invalidated).

CONCLUDING REMARKS

In this short paper, we attempt to show that cost-effective ways of improving database computer performance do *not* lie in the single backend system. Instead, the ways may be found either in multibackend systems or in disk systems with parallel read/write and processing capability. Either way is within the state of the art of hardware and software technologies. What we must do is try these cost-effective ways.

ACKNOWLEDGMENTS

The work reported herein is supported by the Office of Naval Research under Contract N00014–75–C–0573 and, in part, by an equipment grant from the Digital Equipment Corporation.

REFERENCES

1. Canady, R.H., R.D. Harrison, E.L. Ivie, J.L. Ryder, and L.A. Wehr. "A Backend Computer for Database Management." *Communications of the ACM*, 17 (1974), 575–582.
2. "Ampex Parallel Transfer Disk Drive," (DM-PTD9) product announcement, Ampex Corp., Redwood City, Calif., 1977.
3. Banerjee, J., D.K. Hsiao, and K. Kannan. "DBC—A Database Computer for Very Large Databases." *IEEE Transactions on Computers*, C-28 (1979), 414–429.
4. Hsiao, D.K., et al. "The Implementation of a Multi-backend Database System (MDBS): Part I—An Exercise in Database Software Engineering." In *Advanced Database Machine Architecture*. New York: Prentice-Hall, 1983.
5. He, S.G., et al. "The Implementation of a Multi-Backend Database System (MDBS): Part II—The Design of a Prototype MDBS." In *Advanced Database Machine Architecture*. New York: Prentice-Hall, 1983.

Application of the massively parallel processor to database management systems

by EDWARD W. DAVIS
North Carolina State University
Raleigh, North Carolina

ABSTRACT

The Goodyear massively parallel processor (MPP) represents a new architecture with the potential for providing improved solutions to applications benefiting from highly parallel operation. In this paper, application of the MPP to database management systems is examined. Specifically, the relational database model is considered. Database management has been selected as a candidate application of the MPP because of the positive results achieved in previous work related to parallel architectures and database systems. The relational model has been selected for its applicability to parallel processing, its mathematical foundation, and its general recognition as a model that is superior in many respects to the hierarchical and network models. The paper concentrates on a comparative evaluation of the MPP and an abstract conventional computer by examining specific database management functions rather than an entire database management system.

INTRODUCTION

New digital computer architectures offer the potential for finding improved solutions to computer-based applications. The massively parallel processor (MPP) represents a new architecture with such potential.[1,2,3,4] In contrast to many architectural ideas, it is important to note that a physical realization of the MPP is being produced under a contract between NASA and Goodyear Aerospace Corporation.

The MPP is motivated by the necessity to process very large amounts of two-dimensional image data, a problem approaching intractability on conventional machines. MPP system design supports the processing, interprocessor communication, memory, and input/output bandwidth requirements of high-performance image processing. Since the machine is fully programmable, it is reasonable to examine additional applications for which the highly parallel properties are potentially advantageous. That is, it is reasonable to determine applications where the MPP can provide better solutions than those currently available.

In this paper, application of the MPP to database management systems is examined. Specifically, the relational database model is considered. Database management has been selected as a candidate application of the MPP because of the positive results achieved in previous work related to parallel architectures and database systems.[5,6,7] The relational model has been selected for its applicability to parallel processing, its mathematical foundation, and its general recognition as a model that is superior in many respects to the hierarchical and network models.[8,9] The paper concentrates on a comparative evaluation of the MPP and an abstract conventional computer by examining specific database management functions rather than an entire database management system. This is consistent with the commonly held view of the parallel array portion of the MPP as handling certain functions on receipt of a command, whereas the MPP control processor or a host machine generates commands.

MPP FEATURES

Relational database systems can be implemented on conventional uniprocessor computing equipment. They do not require any unique functional capability. However, database system performance can be affected greatly by features of the underlying computer architecture. Thus several machines have been designed specifically for database applications but few have been built. Three design examples are RAP,[6] DBC,[10] and DIRECT.[11] The MPP design, although motivated by image processing rather than by database tasks, nevertheless has features that nicely support the relational database model. Essentially, the massive parallelism can be used to great advantage for fundamental relational operations.

System Organization

Major components of the MPP, as shown in Figure 1, are (1) a program and data management unit, (2) an array control unit, (3) an array unit, (4) a staging memory, and (5) peripheral devices. Processing parallelism occurs in the array unit, a 128×128 array of processing elements (PEs). I/O parallelism is supported by the staging memory, its input and output ports, and an I/O plane formed by a register in each processing element.

Overall supervision of the MPP is exercised by the program and data management unit. This unit provides an external interface to the MPP, responds to high-level commands for MPP use, and initiates activity in the array control unit.

A significant architectural feature of the MPP is the incorporation of three distinct control units and two control memories into a single major component called an array control unit. Application programmers develop software for main control and I/O control units. The software for both resides in main control memory. Systems engineers and programmers develop microprograms for the processing element control unit, which executes out of PE control memory. Main control directly executes scalar operations and passes requests for parallel array operations to the PE control unit via a request queue. Scalar and array operations can proceed simultaneously.

Massive parallelism occurs in the array unit with 16,384 PEs capable of operating in parallel. Each PE is a bit-serial processor. All PE operations, including memory access, take place at a 10-MHz clock rate for a 100-nanosecond operation time. Each PE has a bit-wide random access memory (RAM) with a 16-bit address space. The first MPP will provide 1,024 bits of RAM per PE, expandable to the design limit of 64K, corresponding to 16 address bits. In addition to PE interconnection as an array, a separate unidirectional path allows I/O data movement along all 128 array rows in parallel. Data movement on the I/O path is independent of other PE operations and can occur simultaneously with processing.

Random access memory associated with each PE in the array unit represents the primary level of memory hierarchy. The staging memory, a level in the memory hierarchy, provides both data storage and certain permutations useful in accessing multidimensional data structures. The staging memory is also an integral part of the MPP I/O system for array data. The interface between the staging memory and the array unit is a 128-bit-wide path that also operates at 10 MHz. That is, I/O bandwidth is 160 megabytes per second.

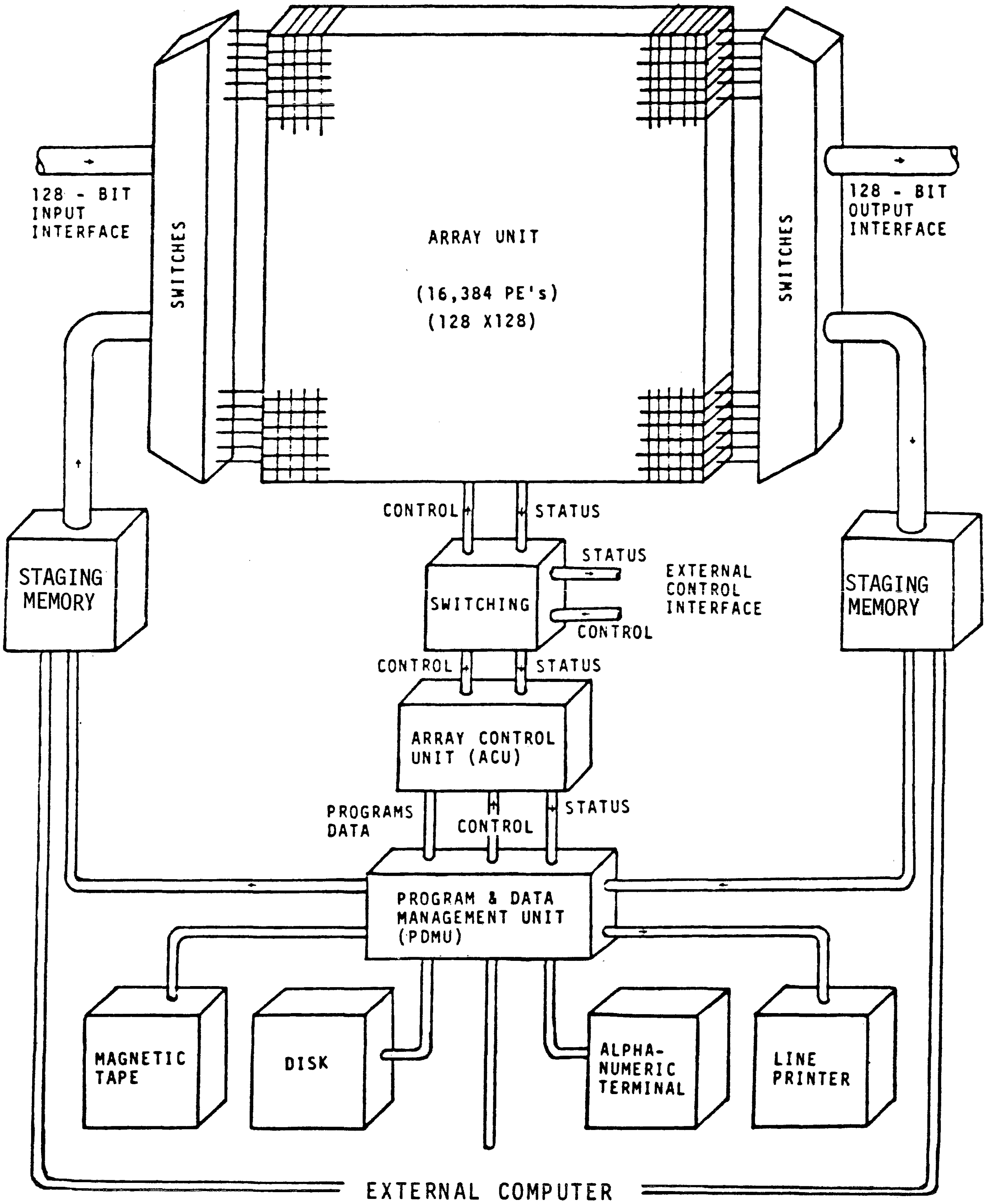


Figure 1—Block diagram of the MPP[2]

System Operation

Processing in the MPP array unit takes place in SIMD fashion. That is, a single instruction stream is executed on multiple (parallel) data streams. All enabled processing elements execute the current instruction. Processing is enabled or disabled within each PE according to the state of a mask register. Masking allows selection of a subset of PEs that will take part in further processing, a feature that is very useful in query and update functions on databases.

Processing elements and their memories are one-bit-wide units; thus processing takes place in a bit-serial fashion. Since each PE in the entire array can operate simultaneously, a 128 × 128 plane of bits can be processed in parallel. A bit plane is simply one bit from the same location in each PE. Processing speed derives from the high degree of parallelism. If an instruction requires a data memory access, each PE accesses the same address in its own RAM. Thus the data streams to be processed in parallel must be mapped onto addresses in parallel memories.

Highly parallel array unit I/O data movement involves I/O ports of the staging memory and a shift register I/O bit plane in the array. A single bit register in each PE is interconnected to form 128 row-oriented right-shift registers. The staging memory presents 128 bits of data at the leftmost edge, one bit per row, for input via shifting. Repeating the input shift step 128 times results in a bit plane with an input data bit at each PE. One write operation can transfer the plane into random access memory.

Output proceeds by reading memory to load the I/O bit plane, then shifting to the staging memory port. Input and output can occur simultaneously in the plane since data shifting in on the left fills columns vacated by data shifting to the right for output.

RELATIONAL DATABASES IN THE MPP

The Relational Model

An important feature of the relational model of data is that the user's view involves just one data structure: the relation. Essentially a tabular structure, a relation consists of rows called tuples and columns called attributes. A database consists of one or more relations. Operations on the data structures result in modification of attribute values or construction of new relations. In other terms, the operations on relations include queries and updates. A query is a search of attributes to identify or select certain tuples. Update operations can include modification of values as well as tuple insertion or deletion.

Figure 2 is an example database in relational form. It is adapted from the Date text.[9] This small example is used to illustrate the tabular nature. The database concerns suppliers of parts. Relation S contains information about suppliers, relation P contains information about parts, and SP provides an association between suppliers and parts based on supplier numbers and part numbers.

S:	S#	SNAME	STATUS	SCITY
	S1	SMITH	20	LONDON
	S2	JONES	10	PARIS
	S3	BLAKE	30	PARIS
	S4	CLARK	20	LONDON
	S5	ADAMS	30	ATHENS

P:	P#	PNAME	COLOR	WT	PCITY
	P1	NUT	RED	12	LONDON
	P2	BOLT	GREEN	17	PARIS
	P3	SCREW	BLUE	17	ROME
	P4	SCREW	RED	17	LONDON
	P5	CAM	BLUE	12	PARIS
	P6	COG	RED	19	LONDON

SP:	S#	P#	QTY
	S1	P1	300
	S1	P2	200
	S1	P3	400
	S1	P4	200
	S1	P5	100
	S1	P6	100
	S2	P1	300
	S2	P2	400
	S3	P2	200
	S4	P2	200
	S4	P4	300
	S4	P5	400

Figure 2—A relational database

Relations in MPP Memory

Processing parallelism in the MPP occurs when an operation takes place on a bit plane distributed over many processors. That is, the same operation is performed on many bits of data. Typically this will be a single bit of data from each of many data items. Data must be accessible in parallel to be processed in parallel.

Database queries involve operations on attribute fields of tuples in relations. Since a given operation occurs on the same attribute of many tuples, parallel processing potential is great. The structure of data in the MPP can allow exploitation of this potential. A relation stored with one tuple per PE has a parallelism factor limited only by the number of tuples in the relation (its cardinality) or the number of PEs. In the MPP, the 16,384 PEs represent the hardware limit on parallelism.

A good data structure for a relation will have an individual tuple stored in an individual PE memory. Such a mapping places corresponding attributes of tuples in corresponding memory addresses for parallel, bit-plane accessing. If tuple length exceeds the realized memory size, a single tuple can be mapped into more than one PE memory. Alternatively, selected attribute fields, rather than the entire tuple, can be stored in PE memory.

MPP Memory Hierarchy

Recall that primary data memory capacity in the MPP ranges from 2 Mbytes in the first realization to 128 Mbytes as a design limit. Primary memory is supported by a hierarchy of memory components. Next to primary memory, the staging memory provides up to 64 Mbytes of solid state memory with a 160-megabytes-per-second access rate. It is designed to buffer data to and from the next level in the hierarchy. As a second major function it can be used to reformat data. The staging memory is designed to be useful in handling access to rows and columns of multidimensional data structures. For example, it can provide columns of attributes as needed for array unit I/O, and it can provide rows of tuples as may be needed for the host or a peripheral.

The next level in the memory hierarchy are devices peripheral to the MPP, such as a disk memory system. All levels of the hierarchy provide physical storage for relations that is very similar to the user's logical view. On disks, a relation is a file with tuples as records. The close fit between the user's view of relations and their physical storage simplifies the conceptual design of a relational database system for the MPP. Software to manage the data and to access it is correspondingly simplified.

DATABASE OPERATIONS

This section describes implementation of selected database operations in the MPP and provides a performance comparison of the MPP and an abstract conventional sequential processor (CSP). Processing operations on data in primary memory are considered as is treatment of larger databases using I/O and the memory hierarchy. Performance benefits related to the parallel architecture are emphasized. Relative dollar cost is not considered since the comparison involves an abstract conventional processor and a developmental parallel processor.

Certain assumptions are made to provide a basis for performance comparisons. Both the MPP and the conventional machine are assumed to have the same execution-cycle time. In the MPP an execution cycle is a bit-level operation in each PE. In the conventional machine a word-level operation occurs, regardless of the word length. It is also assumed that primary memory available for data storage is equivalent in each machine. That is, the conventional machine has 2 Mbytes of primary memory, in addition to that used for the operating system and programs, to match the MPP minimum configuration.

Query Operations

The implementation and performance of queries on a database resident within PE array memory is described. MPP primary memory, that solid-state random access memory directly connected to the PEs, is capable of containing a moderately sized database. Using 1K-bit chips as in the first realization provides 16 Mbits of data memory. At the design limit 64 Kbits per PE, memory capacity exceeds 10^9 bits. Although a few bit planes are reserved for system use, none of the space is used for program storage.

Selection of tuples that satisfy a query corresponds to the select operation in relational terminology. The result of such operations, a horizontal subset of the original relation, is indicated by a flag bit in each PE. Further processing could involve retrieving information from selected tuples, counting responders, and so on.

A query on a single attribute of one relation is the simplest query. It is implemented as a comparison of a single comparand and the attribute field of all tuples. MPP design supports this operation as an instruction at the application programmer's level. Using the database from Figure 2 as an example, suppose the query is to find all suppliers located in London. The single instruction to achieve this is:

```
EQSA,S     'LONDON', S.SCITY, RESULT
```

Interpretation of the instruction is to perform an equality comparison of a scalar and an array in PEs enabled by logical '1' in the relation S flag bit. The scalar value is 'LONDON'. The vector is the field in memory defined for relation S and attribute field SCITY. PEs with tuples satisfying the search should set the bit symbolically named RESULT.

Within the array, at the microprogram level, the operation is a loop whose iteration count depends on attribute-field bit length. Response time is proportional to attribute-field length but independent of the number of tuples. In a conventional sequential processor (CSP) the operation also involves looping. Since comparisons occur sequentially, the iteration count is dependent on the number of tuples. CSP response time is proportional to the number of tuples but independent of attribute-field length.

Based on items that determine query response time, a first-order performance evaluation is that the MPP will be faster than the CSP whenever the number of tuples in a relation exceeds the number of bits in the attribute field. Figure 3 is a graph of relative query timing on a log-log scale for a 32-bit

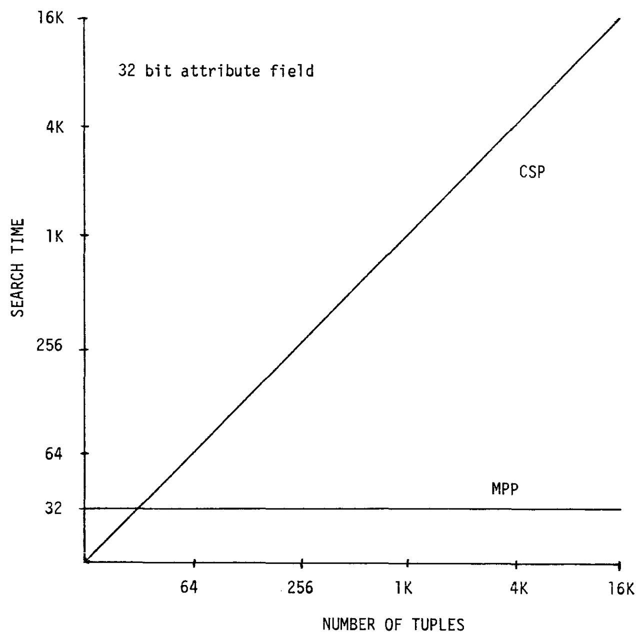


Figure 3—Simple query timing comparison

attribute-field length. It shows the significant performance difference resulting from increasing parallelism as the number of tuples increases.

A query involving multiple attributes of one relation may be expressed either as a logical combination of single-attribute queries or as a comparison of pairs of attribute fields. Logical combinations can be readily achieved in the MPP via logical operations among bit planes resulting from single-attribute queries. A comparison of pairs of attribute fields within a tuple, for all tuples, can be characterized as a comparison of corresponding elements of two single-dimension arrays. The MPP application program needs just one instruction to carry out such an array-to-array search. Execution time is proportional to attribute-field length.

Queries discussed up to now were limited to a single relation. Query responses could be totally determined from information in the relation. Databases typically use several relations, making it necessary to be able to combine information in the operation known as join. This is a more complex query for both MPP and CSP machines.

Using the database from Figure 2 as an example, suppose the query is to find all suppliers of bolts. The supplier relation (S) does not specify what is supplied. The parts relation (P) does not specify the supplier of each part. The supplier-part relation (SP) associates supplier and part numbers but does not identify the type of part. An outline of a method used in the MPP to arrive at the result is:

```
Search P.PNAME for 'BOLT', flagging all responders.
While responders remain do:
    Retrieve one responder's P.P# and reset its flag.
    Search SP.P# for the retrieved P.P# value.
    Logically OR result into a result bit plane.
End.
```

In this method both parallel and sequential activities are used. Searches are highly parallel. Resolving responders and retrieving one can be done with a minimum value search on P.P# and some sequential scalar operations. Iterations of the while-do loop occur sequentially. A very similar method can be used in a CSP. It is necessary to sequentially search all tuples of P.PNAME for 'BOLT', followed by a search of SP for each tuple representing a bolt.

Timing considerations show again that MPP performance is proportional to attribute-field lengths while CSP performance is proportional to the number of tuples. For large databases, the number of tuples will certainly be much greater than attribute-field lengths, yielding longer operation time for the CSP. That is, the MPP demonstrates a large performance increase for the join operation.

Result Retrieval

Ultimately, it will be necessary to retrieve results of queries. When an attribute value from a relatively large number of tuples must be output, the I/O plane provides a parallel, high bandwidth path. The method is to move one bit of the attribute from each selected tuple out of the PE array in parallel, then to iterate the process through all bits of the attribute. Output of the bit requires a PE memory access to load the I/O plane register, followed by right shifting out of the array and into a staging memory port. Shift count depends on the position of selected tuples in the PE array. If all selected tuples are located in the rightmost column, a shift of one completes output. In the worst case, a selected tuple may be located in the leftmost column requiring a full 128-position shift. Output time, at 100 nsec per shift, is thus based on the column position of tuples selected for output and on the number of bits in the field.

Using the I/O plane method just described, output time is independent of the number of selected tuples in the columns being output. Thus, while output of an n-bit attribute from one tuple can take as long as 12.8 n μsec, many more attribute values could be output in the same time. An effective output rate can be determined for an assumed percentage of selected tuples. Figure 4 shows the effective rate under the assumption that result attributes are distributed over all PE columns, requiring a full 128-position shift. When results are present in 3% of all tuples, for example, the effective 32-bit attribute output time is an excellent 0.8 μsec.

When an attribute value from one or a few tuples must be output, the use of special sum-OR logic is better than the I/O plane. Figure 4 shows an increase in effective output time using the I/O plane method as the percent of potential output items decreases. The parallel register output paths are not efficient when the parallelism factor is small. Alternatively, sum-OR logic provides an efficient path from one enabled PE to a scalar register in the PE control unit, which is accessible by the main control unit.

Update Operations

Databases are dynamic. The capability to change the information in a database by adding new information, removing that which is no longer needed, and changing existing information is necessary. This section is concerned with the dynamic nature of a database.

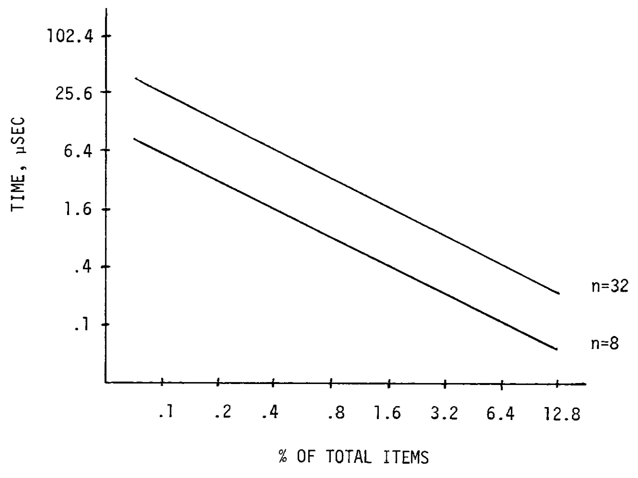


Figure 4—Effective attribute output timing

Increasing the size of a relation by inserting tuples is a straightforward process in the MPP. The definition of a relation is a set of tuples with no implied order of set members. Query processing in the MPP also does not depend on an ordering. Again, the user's logical view of a relation matches the MPP physical implementation. A tuple can be inserted by simply appending it to the existing relation. A flag bit in each PE memory can indicate PEs with tuples belonging to a given relation. The use of flag bits to enable PEs via the mask register and the ability to handle unordered relations reduces complexity of the system design and software.

Parallelism, as discussed for query processing, can also be used for modifying attributes of tuples. The MPP is capable of performing arithmetic and logical operations on data in all enabled PEs in parallel. Modification of an attribute field by a single value is similar to a simple query on an attribute. It is a scalar-to-array operation. Timing relationships are similar to the query case with MPP timing proportional to the field length in bits and CSP timing proportional to the number of tuples to be modified. Functional operations between fields of selected tuples in a relation are implemented as array-to-array operations. Corresponding elements in each array are functionally combined for all tuples simultaneously. The processing is bit serial but tuple parallel.

Very Large Databases

Prior discussion centered on a primary-memory resident database. For larger databases, I/O involved in moving the database between secondary memory and the PE array must be considered.

In the MPP, processing and I/O can be almost totally overlapped in time. Processing uses data paths that are independent of those used for I/O. When the staging memory is operating at its normal transfer rate, 100 nsec per 128-bit column, a 16K-bit plane can be input in 12.9 μsec. The time is based on 128 shifts and one write, each taking 100 nsec. For an n-bit field, input time is $(n)(12.9\ \mu\text{sec})$. The given time holds for any number of tuples up to the 16K PE array size. In excess of 16K, the input process must be repeated once for each integral multiple of the 16K size. Thus, MPP input time for a set of r tuples of length n bits is:

$$t_{\text{MPP}} = (n)(12.9\ \mu\text{sec})\ (\text{ceiling}(r/16384))$$

To arrive at a time comparison, input time in a CSP is similarly defined. Assume input occurs at a 100-nsec rate for each 32-bit item. For an r-tuple relation, input time is $(r)(0.1\ \mu\text{sec})$. The given time holds for any number of bits in the field length up to the 32-bit memory-access size. In excess of 32 bits, the input process must be repeated. Thus CSP input time for r tuples of length n is:

$$t_{\text{CSP}} = (r)(0.1\ \mu\text{sec})(\text{ceiling}(n/32))$$

If the time expressions for both machines are equated, values of r and n that satisfy the equation represent the breakeven graph plotted in Figure 5. This figure shows that MPP

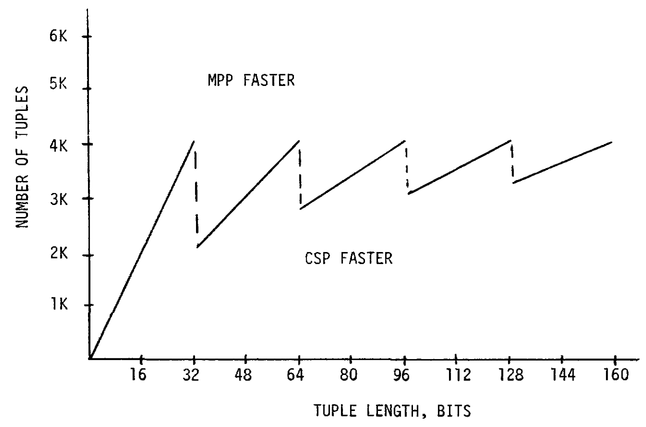


Figure 5—Relation input timing breakeven dimensions

input is faster than CSP input, regardless of tuple length, when the number of tuples exceeds 4,128. MPP input performance is obviously well suited for large relations.

Emphasis in this section has been on input data movement. However, because of I/O plane and staging-memory design, output can occur simultaneously. The only difference from a pure input process is that one memory access is used to load the I/O plane with output data prior to the input actions. Right shifting to input data from the left simultaneously outputs data on the right. Both input and output are between the array unit and the staging memory. The additional memory access used for output adds 100 nsec to I/O time. That is, one bit plane can be output and a different plane input in a total time of 13.0 μsec. This amount of time per bit plane allows complex queries or functional processing to take place on data in the PE array.

CONCLUSION

This paper shows that MPP performance is better than a high-performance conventional sequential processor for relational database query and update operations. Parallelism in the MPP architecture results in processing and I/O speed. It also provides a direct model of the tabular data structure used in relational systems. Close correspondence of the implementer's logical view of a database with the machine's physical structure will simplify database system design and software.

ACKNOWLEDGMENT

The cooperation and support of Goodyear Aerospace Corporation in the conduct of this work is appreciated.

REFERENCES

1. Fung, L-W. "A Massively Parallel Processing Computer." In D. J. Kuck et al. (eds.), *Proceedings of the Symposium on High Speed Computer and Algorithm Organization.* New York: Academic Press, 1977, pp. 203–204.
2. Batcher, K. E. "Design of a Massively Parallel Processor." *IEEE Transactions on Computers,* C-29 (1980), pp. 336–340.

3. Tsoras, J. "The Massively Parallel Processor (MPP)—Innovation in High Speed Processors." *Proceedings of the AIAA Computers in Aerospace III Conference,* October 1981.
4. Batcher, K. E. "Bit-Serial Parallel Processing Systems." *IEEE Transactions on Computers,* C-31 (1982), pp. 377–384.
5. Moulder, R. "An Implementation of a Data Management System on an Associative Processor." *AFIPS Proceedings of the National Computer Conference,* (Vol. 42), 1973, pp. 171–176.
6. Ozkarahan, E. A., S. A. Schuster, and K. C. Sevcik. "Performance Evaluation of a Relational Associative Processor." *ACM Transactions on Database Systems,* 2 (1977), pp. 175–195.
7. Capraro, G. T., and P. B. Berra. "A Data Base Management Modeling Technique and Special Function Hardware Architecture," TR-79-14, Rome Air Development Center, Griffiss AFB, New York, January 1979.
8. Babb, E. "Implementing a Relational Database by Means of Specialized Hardware." *ACM Transactions on Database Systems,* 4 (1979), pp. 1–29.
9. Date, C. J. *An Introduction to Database Systems* (3rd ed.). Reading, Mass.: Addison-Wesley, 1981.
10. Banerjee, J., D. K. Hsiao, and R. I. Baum. "Concepts and Capabilities of a Database Computer." *ACM Transactions on Database Systems,* 4 (1979), pp. 1–29.
11. DeWitt, D. J. "DIRECT—A Multiprocessor Organization for Supporting Relational Database Management Systems." *IEEE Transactions on Computers,* 28 (1979), pp. 395–406.

Panacea or pitfall? The impact of relational databases on your environment

by WILLEM STOELLER
Arthur Andersen & Co.
Chicago, Illinois

ABSTRACT

This paper discusses the impact of relational database management systems (DBMSs) on systems development. It states conditions and characteristics for which relational DBMSs are most applicable and appropriate. Finally, it suggests improvements in relational DBMSs in the areas of performance, data integrity, standardization, and a user-friendly interface.

Appendixes define relational system terms and compare DBMSs with two other data models.

INTRODUCTION

The use of database management systems (DBMSs) during the last five years has increased at a significant rate. A DBMS provides users with a more controlled and flexible environment than that offered by the basic access methods within the operating system. Since DBMSs are the caretakers and delivery mechanisms for data, their evolution will affect all information-processing disciplines and business areas. Relational DBMSs, commercially available since around 1978, could play a dramatic role within the DBMS area. These are the only DBMSs based on a theoretical data model (the relational model, which was developed by E. F. Codd in the early 1970s). Because of its mathematical foundation, the relational model's simple architecture can accommodate new features without the sharp increase in complexity common in most software systems. The most well-known products on the market today are INGRESS, NOMAD, ORACLE, QBE, and SQL/DS. Some are basic data management systems, while others offer additional facilities such as screen painting, report writing, and graphics. Implementations for the business environment, however, have lagged, primarily because of fear of poor performance. However, the current industry emphasis is on management systems that use query-oriented or end-user "fourth generation" data manipulation languages rather than on transaction systems. This trend has pushed relational systems into the spotlight and made relational a popular buzzword in the trade press and in vendor marketing strategies.

This presentation outlines the potential impact of these relational products on your application development and operations. Further, it presents suggestions for improvement of these products.

KEY ISSUES

How will the impact of relational databases relate to the key issues in your environment?

The major issues in the DBMS area include

1. The large applications backlog. Estimates set the average backlog at between two and ten years.
2. The quality of many existing applications and related databases. Current development techniques are ineffective and inefficient in meeting the end-user needs.
3. The inflexible data structures of traditional databases and lack of logical data independence of applications. Changing data structures of existing databases is costly and time consuming.
4. The difficulty of proper physical database design. Database design for products such as IMS and IDMS is complex and costly to adjust. Highly skilled people are required.
5. The design and testing of applications using traditional databases. Applications that must specify navigation through the database are error prone and hard to maintain.
6. The long development time of applications. Today's complex business environment often tolerates only short lead times for information.

PANACEA OR PITFALL?

Characteristics

The Relational Task Group of the American National Standards Institute developed a set of criteria to define relational database systems. These include the following:

1. All data in the database are represented as values in tables.
2. The data manipulation language does not contain explicit navigation links between tables.
3. The command set allows selection of subsets of tables and joining of data from different tables.

(Appendixes A and B cover some terminology and a comparison of different data models.)

In addition, most relational systems have at least the following components:

1. A data definition and a data manipulation language based on the relational data model.
2. A user-friendly interface with interactive command execution; an editor for entered commands; and help functions to support the user. The commands can be used in several host languages and utilities in addition to the user-friendly interface.

IMPACT ON THE DEVELOPMENT AND OPERATION

Analysis and Design

After initial analysis of the user's information needs, a prototype of the relational database and the application can be developed with the aid of the user-friendly interface.

End users will be more able to learn about their real information requirements. Armed with this feedback, analysts and programmers can adapt the design of the application. The end result is an application and database that more closely meet the users' information requirements with fewer design bugs.

The preparation of program specifications will be simpler since access to the database can be specified without consideration for navigation or the underlying physical design of the database.

Programming

Programmer productivity will be higher because

1. Parts of the application can be created using procedures based on the user-friendly interface.
2. The database sublanguage is set instead of record-oriented and contains very powerful selection capabilities. This will simplify the logic of the host-language part.
3. The data manipulation language contains only four basic facilities: data access, add rows (records), change elements, and delete rows.
4. Navigation does not need to be specified and strong physical and logical data independence are provided.
5. Training will be much faster than it is for the traditional database sublanguages.

Data Administration

Data administration is a particularly critical function with relational systems because of the emphasis on *data* rather than on processing logic. A central coordinator is crucial since different user groups use existing data and create new data for particular business needs. The data administrator must keep track of these developments to ensure that data is used consistently, is properly validated, and is not redundant. The administrator must ensure that end users understand and use good, basic data-analysis techniques. The data administrator must find ways to get information about the database to the interactive user. The end user must have access to the current data structure, validation rules, column domains, and database status.

With relational systems, data definition commands are very straightforward. Any knowledgeable user can easily create or change table structures. Restricting this capability is the only way the administration function can exercise control over the database.

Relational DBMSs offer excellent, easy-to-use tools for administration of data security and privacy, usually on the element and row (record) occurrence level.

The most important aid to effective data administration is the data dictionary. Although most relational systems have no fully integrated dictionary, many of the same features are offered by the system catalogs. (The system catalogs are tables used and updated by the system to keep track of information about the database.)

Database Administration

Database administration is considerably easier with relational systems. In a hierarchial DBMS such as IMS, the database administrator must coordinate multiple databases, select efficient access methods for each, and choose pointers and other parameters to define the access path a user may follow. With relational systems, these tasks are eliminated since the database software will automatically optimize access to the data. Database performance tuning is reduced to defining or dropping indexes for specific tables and requesting clustering of frequently linked tables.

Changes to the data structures are easy to implement without a lengthy unload/reload procedure. Existing applications are often unaffected by data structure changes owing to a high degree of physical and logical data independence.

End Users

Because of the powerful user-friendly interfaces, many non-DP professionals can use most of the data-manipulation commands with a small training investment.

For data definition and update procedures, end users still need the guidance of DP professionals.

APPLICABILITY OF CURRENT PRODUCTS

Relational DBMSs should be considered a viable possibility for applications with the following characteristics:

1. Table sizes are less than 500,000 rows (records) and performance requirements are modest.
2. Transaction volumes are low, even in peak periods.
3. Interactive and/or ad hoc, database queries are fundamental.

The applications also should have one or more of the following characteristics:

1. Data must be accessed dynamically—data requirements and combinations are not known until execution time.
2. Data structure may evolve over time.
3. Data integrity, data security, data privacy, and data sharing are important.
4. Data will be updated on line.
5. Prototyping of database and query functions is needed to properly define complex requests or to communicate effectively with users.

These latter five conditions are indicators of how appropriate the situation may be for usage of current relational products. They do not all need to be true for the relational systems to be a good fit.

Relational systems, as they stand today, have some limitations that make them inappropriate for applications with one or more of these characteristics:

1. Data volume is large (over a million rows) and/or performance requirements critical.
2. Transaction volumes are large.
3. Data structure is naturally hierarchical, with considerable hierarchical reporting expected.
4. Users of interactive query facility will be casual (less than once a week).

SUGGESTED IMPROVEMENTS

Performance

Relational systems can accommodate medium-volume databases, but their performance will be questionable.

Since relational products are all fairly new, further software enhancements will improve performance. Some new storage media, such as content addressable files (CAF from ICL) may provide a solution.

Data Integrity

As with the traditional DBMSs, data integrity relationships between tables are weak in current products. (This is where a row/column value in one table is verified against a required key value in another table.)

Standardization

Standardization of the relational sublanguage will stimulate development and use of relational systems. Currently, relational database standards are being developed by the ANSI Database Standards Committee.

User-Friendly Interface

To appeal to the large community of non-DP professionals, the user-friendly interface needs improvement in the following areas:

1. Syntax. QBE showed that a totally different approach is feasible.
2. Editing. Easy-to-use full-screen editing is desirable.
3. Guidance for interactive users. They should be provided with prompting and menu facilities, clear warnings and help functions, and more forgiving software.

CONCLUSION

Relational systems will claim a sizable database market share in the next five years.

Simple, flexible data structures and supporting English-like command language are what the industry is looking for.

APPENDIX A: RELATIONAL SYSTEM TERMS

Introduction

The area of relational systems has emerged with its own unique language. The mathematical terms reflect the theoretical origins of the relational model. This appendix explains some of the common concepts and terms of relational models.

Relation

A relation is a normalized data aggregate represented by a table. (The terms relation and table can be used interchangeably.) A relation consists of any number of columns in any order and rows in any order. Figure A-1 shows an example of an Employee relation.

Row

Each row is one occurrence of the relation. A row gives a complete piece of information about the relation. No two rows are completely identical, and the unique portion of each row can be in one column or may span several columns. A row (sometimes called a tuple) is similar to a record in a file. The Employee relation in Figure A-1 contains two rows. The meaning of a relation does not change when rows are added or deleted.

EMPLOYEE:	*NUMBER*	*NAME*	*SALARY*	*DEPT*
	61256	Jones	8000	Household
	38972	Anderson	6000	Toy

Figure A-1—Employee table

Column

Each column is a single data item and represents an attribute of the relation. Repeating groups of data items is not allowed within a column or be repeating columns. A column is similar to the name of a field on a record. The Employee relation in Figure A-1 contains four columns: NUMBER, NAME, SALARY, and DEPT. The meaning of a relation changes when columns are added or deleted.

Domain

The collection of all possible allowable values for each column is called its domain. The domain of the NAME column in the Employee relation is the last name of all employees.

Relational Database

A relational database is simply a collection of relations. The relational database used as an example for the relational operations discussed here.

EMPLOYEE:	*NUMBER*	*NAME*	*SALARY*	*DEPT*
	61256	Jones	8000	Household
	38972	Anderson	6000	Toy
	09181	Morgan	10000	Cosmetics
	22318	Murphy	9000	Toy

MANAGER:	*DEPT*	*MANAGER*
	Toy	Murphy
	Household	Smith
	Cosmetics	Hoffman

Figure A-2—Manager table

Relational Operations

Relational operations access and manipulate the relations of a relational database. These operations are nonprocedural since they define what to do but not how to do it. In a relational system, data can be accessed only by matching data values. The database user defines what values to match and the system determines the access path to the desired data.

Relational operators work on entire tables (rather than on individual records) and result in new relations. The three principal operations are

1. Project
2. Select
3. Join

Project

The project operation extracts *columns* from a relation to create a new relation. Duplicate rows are dropped. The following list demonstrates the project operation where the NAME and SALARY columns were selected from the Employee relation in Figure A-2.

NAME	*SALARY*
Jones	8000
Anderson	6000
Morgan	10000
Murphy	9000

Select

The select operation extracts from a relation to create a new relation. The database user defines the criteria to use in the selection by describing desired data values. The example in Figure A-3 shows a select operation on the Employee relation where only employees in the Toy Department are selected.

NUMBER	*NAME*	*SALARY*	*DEPT*
38972	Anderson	6000	Toy
22318	Murphy	9000	Toy

Figure A-3—Example of a selection

Join

The join operation combines two or more relations to create a new relation. One column in each table must share a common domain to make the join meaningful. Figure A-4 shows an example where the Employee and Manager relations from Figure A-2 are joined on the basis of the DEPT column in each table.

NUMBER	*NAME*	*SALARY*	*DEPT*	*MGR*
61256	Jones	8000	Household	Smith
38972	Anderson	6000	Toy	Murphy
09181	Morgan	10000	Cosmetics	Hoffman
22318	Murphy	9000	Toy	Murphy

Figure A-4—Example of a join

Other Terms

An *element* is a single field value in a relation. It is the intersection of a row and column.

The *degree* of a relation is the number of columns it has. An N-*ary relation* is one with N columns. A relation with two columns is a *binary relation*.

The number of rows in a relation is called its *cardinality*.

A row or record occurrence is sometimes called a *tuple*. An *N-tuple* is a row from a relation of N columns.

APPENDIX B: COMPARISON WITH OTHER DATA MODELS

Introduction

A database usually fits one of the three data models: hierarchical, network, or relational. This appendix compares the relational model to the hierarchical and network models.

Access Path

In a hierarchical or network data model, the connections and relationships between data aggregates remain in the data structure. The data designer must carefully predict all data access to ensure that needed data can be extracted via some access path. Two techniques used in that process are to identify alternate entry points and to use unidirectional business-function data models.

With relational systems, access paths are not predetermined. The relationships between data aggregates are considered in design but are not implemented in the database. Since join operations simply combine tables, the direction of access is irrelevant.

This flexibility in accessing data is the main reason why relational systems excel in unplanned (ad hoc) data requests.

Navigation

The hierarchical or network database user must enter the database at an acceptable entry point and then navigate from aggregate to aggregate through the structure to get the needed data. This usually requires the data manipulation language to be coded in a language such as COBOL and embedded in program logic. Conventional programs are needed just to provide the logic to navigate to the data.

Relational database users can manipulate each relation independently or can join relations together as needed. Because the data can be directly accessed, the relational model lends itself to fourth-generation languages. The access logic is in the DBMS, not in the application.

Level of Access

The data manipulation languages for hierarchical and network systems generally return one record at a time. Therefore, the application logic must interact frequently with the DBMS to obtain multiple records.

Relational operations manipulate entire relations in a single request to the DBMS. The relational system always creates a new relation as a result of an operation. The fact that table operations create tables is called the closure property.

Maintenance

Even a small change to a data aggregate in a hierarchical or network system may have a significant impact on the physical implementation and application programs. Programs may need to be recompiled, database definition utilities rerun, and the entire database reloaded. This maintenance is almost always performed via batch processing.

Relational systems generally allow new relations to be defined interactively and new columns to be added to existing tables. Since columns do not have any inherent sequence, each operation can order the columns in any sequence.

Physical Database

Hierarchical and network systems use indexes, required pointers, and key fields to keep occurrences of an aggregate in sequence. Physical database design uses the concept of physical proximity of related records to improve performance. Application program logic must know about and properly use indexes to support processing and database structuring.

Indexes are never required in a relational system. Rows are not physically stored in any particular order, so the sequence of occurrences cannot be used to store information about the rows or their relationship. Rows may be sequenced when a relational operation is used. Physical design is shielded from the database user.

Language

Hierarchical and network systems have a data-manipulation language to access data and a distinct data-definition language to create the data structures. The data-definition process typically consists of a set of control cards that are input to a batch job, and the data manipulation syntax is programmer oriented and embedded in application programs.

In contrast, the data-definition and data-manipulation languages for a relational system are one and the same. The syntax is end-user oriented and can be used either interactively or embedded in source programs.

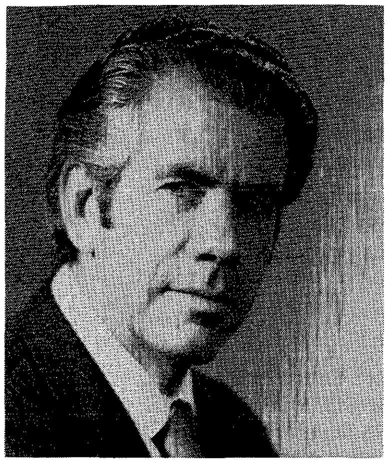
Conclusion

Relational systems separate logical data design from physical design. This data independence gives relational systems more flexibility in accessing data, allows user of higher-level languages, and lessens the impact of database changes. Hierarchical and network systems provide less data independence, so changes to the physical database affect the logical data design.

BIBLIOGRAPHY

1. Date, C. J. *An Introduction to Data Base Systems* (3rd ed.). Reading, Mass.: Addison-Wesley, 1981.
2. Date, C. J. *An Introduction ot Data Base Systems, Volume 2.* Reading, Mass.: Addison-Wesley, 1982.
3. Advanced Computer Techniques Corporation. *Advanced Computer Techniques/Industry Measures* Vol. 3, No. 2, March/April 1981.
4. Codd, E. F. "SQL/DS What It Means." *Computerworld,* 16 March 1981.
5. Kim, W. "Relational Database Systems." *Computing Surveys,* 11 (1979).
6. Canning Publications. "Relational Data Systems Are Here!" *EDP Analyzer,* October 1982.
7. Sanberg, G. "A Primer on Relational Data Base Concepts." *IBM Systems Journal,* 20 (1981).
8. Special Report: "Real Benefits of Data Base Technology," Computerworld, October 25, 1982.

HUMAN AND SOCIAL ISSUES



David L. Holzman
Holzman & Associates, Inc.
Manhattan Beach, California

For the first time, the National Computer Conference devotes an entire program track to the human, organizational, and social effects of computers. This provides an opportunity to bring together an outstanding group of speakers, many of whom have spent years studying the computer's impact on work, organizations, human behavior, and society in general. The theme running through their presentations is that by understanding and responding to these impacts, systems, organizations, and society become more effective.

The program begins with "Organizations, Information Systems, and Office Effectiveness." The panel will present the results of two recent National Science Foundation studies on the use of organizational design models to insure effective implementation and adaptation of systems. Their findings support the use of organizational redesign as a tool for increasing systems' effectiveness.

The next panel takes a more futuristic and international view in exploring the impact of "Fifth Generation Computers." The panel will assess Japan's recently announced projects and present U.S. industry leaders' plans for building up our technology base for the 1990s. They will discuss why, when, how, by whom, and how much.

The panel "Managing Computer Change" brings together an experienced group of panelists who will address the problems of change and uncertainty between managers and the system, managers and their personnel, and system developers and users. The speakers will give illustrations of how system developers and managers must appear not to manipulate while they help the users to deal with uncertainty.

The next panel, "Measuring the Impact of Information Techniques," will discuss and present case studies on the effect of computers in the office environment. They will go beyond the narrow "productivity" measures to throw light on the broader organizational impacts and how they can be measured.

For the panel "Living with Computers: The Multi-Societal Effects," the panelists will present current research findings on socioeconomic impacts within the computer industry, organizational impacts within local government, and impacts on individual behavior within organizations.

"Computerized Society—Resilient or Vulnerable?" will discuss whether a new and serious vulnerability to disruptions or computer unavailability exists, or whether the society has an intrinsic resiliency to major disruptions.

The ergonomic and human factor issues are discussed in the "Man-Machine Interaction" panel. The presentations will cover current ergonomic problems and their solutions, plus improved interface aids to mainframe IBM systems.

The panel "Computer Aids to the Handicapped" presents three applications: (1) sensory and information processing aids for the blind; (2) communications aids for the nonvocal; and (3) a lip-reading aid for the deaf. The applications' possible wider use in office automation systems will be explored.

The final panel is titled "The Institutional Dimensions of Computing in Organizations and Society"; it will discuss dimensions such as persistence and rigidity that can be introduced by computers. The pros and cons of such dimensions will be explored in organizations and in the larger society, including the Soviet Union.

Advanced office systems: An empirical look at use and satisfaction

by T. K. BIKSON and B. A. GUTEK
The Rand Corporation
Santa Monica, California

ABSTRACT

Preliminary research findings from a study of advanced office systems in varied user contexts are summarized here. The research, funded by the National Science Foundation Productivity Improvement Research Section, is intended to determine how information technology is successfully incorporated into white-collar work. Participating in the study were 55 offices. They ranged in size from four to 37 employees and represented 26 different organizations that ranged in size from a total of eight employees to multinational operations. The data reported here come from questionnaires administered to 530 employees in the 55 offices.

Findings reported here address several issues relevant to implementing advanced office systems: (1) White-collar office employees can be classified into four types: management and administration, data-oriented professionals, text-oriented professionals, and support staff. (2) White-collar work forms a systematic cluster of information-handling activities, some of which are performed by nearly everyone in the office. (3) A large percentage of employees, including senior managers and professionals, already use computers in their work; and most nonusers expect to use them in the near future. (4) Four aspects of computer systems underlie user satisfaction: functionality, equipment performance, interaction features, and office environments. (5) Satisfaction with functionality is the best predictor of use of the system. (6) The most important organizational influences on use of and satisfaction with information technology are variety in work and the organization's approach to technological change.

INTRODUCTION

Preliminary research findings from a study of advanced office systems in varied user contexts are summarized here. The research was funded by the National Science Foundation's Productivity Improvement Research Section; its goal is to identify major factors that determine how successfully information technology is incorporated into white-collar work.*

A general understanding of how advanced office systems affect white-collar work is urgently needed—

1. White-collar employees, who already constitute more than 50% of the nation's labor force, are its most rapidly growing subpopulation. And within any organization, whether product or service oriented, "office work" is highly labor intensive and relatively costly.
2. Of the estimated 3.5 million offices in the U.S., about 1.5 million are currently large enough for some sort of advanced information system. That figure will increase as small, flexible systems become more available and entry costs decline.
3. Although computer technology is viewed as a way to improve productivity and decrease labor costs, very little is known about either the nature of white-collar work or information systems in that context.

Consequently, we undertook a study of how computerized procedures are being introduced in private-sector office settings where users are not computer experts. Our hypothesis was that the sequence of implementation decisions—decisions about the kind of system to introduce and especially about how to introduce it—would have important short-term implications for use and satisfaction, with long-term impact on organizations and their productivity. Our basic research questions were

1. What is the nature of white-collar work, and into what patterns do its tasks fall?
2. Who is using computers in office work, and what kinds of procedures are being computerized?
3. What do users like and not like about their computer systems? Do certain features of the systems and their implementation predict users' reactions?
4. Are there any organizational characteristics that affect the implementation of an information system?

*This note reports research in progress on computerized procedures in office settings supported by NSF Division of Industrial Science and Technological Innovation, Grant Number ISI-8110792, T. K. Bikson, Principal Investigator.

For purposes of this research, office settings were defined as organized white-collar work units of four or more employees. Depending on the firm, these units may be called departments, teams, sections, and the like. But all the units have a *skill mix* that may range from senior executive to entry level clerk, and all have *information handling* as a chief responsibility.

The advanced information technology of interest in this setting is the multifunction interactive computer system deployed to replace or enhance traditional office tasks. By *multifunction* we mean that the system is appropriate for a variety of white-collar tasks; by *interactive* we mean that to some extent the user guides the system's activities online. The technological focus is broad, encompassing both standalone computers and multiple terminals linked to mainframe computers, as well as a number of intermediate options.

To examine the implementation of advanced office systems, we solicited the cooperation of organizations in which such technology had been installed for at least 6 months. In each participating office we administered a survey to all members of the work group, whether or not they worked directly with the computer. The survey inquired about information handling tasks, computer system features and usage, and effects on work. In addition, the survey included a number of standard measures of organization characteristics.

To supplement the survey information, we also collected descriptions of computer equipment and software, and related-staffing and budgeting patterns. Finally we conducted a structured interview with the individuals responsible for managing the work group.

Outcomes reported here represent analyses of survey data. The basic findings are

1. White-collar work can be analyzed in terms of systematic clusters of information-handling activities; some of these tasks are performed by nearly everyone in the organizational hierarchy.
2. White-collar workers are well educated and typically have some keyboarding skill; a substantial minority have had some contact with a computer in prior jobs.
3. A large percent of senior managers and professionals already use computers in their work, and most nonusers expect to use them in the near future.
4. Four summary dimensions underlie users' satisfaction with their systems: functionality, equipment performance, system interaction, and office environment.
5. Users are generally pleased with their system's functionality but give low ratings to the office environment and to some aspects of system interaction such as user manuals.

6. Users believe that the technology has positive effects on the quality of their work and also on their productivity.
7. User satisfaction with functionality is the best predictor of system use; functionality coupled with an adequate office environment constitute the best predictors of overall satisfaction with the new technology.
8. The most important organizational determinants of employees' use of and satisfaction with technology are variety in work and the organization's approach to technological change.

In what follows we discuss the survey and its results in detail.

RESEARCH PARTICIPANTS

Characteristics of our research sample are listed below.

26 organizations

50%	Manufacturing
50%	Service

55 work groups

24%	Management, administration
29%	Professional, text-oriented
20%	Professional, data-oriented
27%	Secretarial, clerical, technical support

530 employees

Fifty-five offices from 26 different organizations participated in the research. The organizations ranged from those with as few as 10 employees to multinational operations. About half the organizations were engaged in manufacturing. The remainder provided a variety of services, such as public information, financial management, research, and banking.

The sample is not a random one; thus, it is *not* representative of office work groups in general. Rather, organizations were selected because they were early adopters of new technology. They either volunteered to participate or agreed to do so when asked. Nevertheless, the sample is diverse and shows the range of office functions that may involve computer use.

The focus of the study is the work group rather than the individual. A work group is a group of four or more employees who share an information-handling function—for example, writing and editing text, or preparing economic forecasts. The group might contain workers from several different occupations—for example, a writer, a manager, and a secretary. A group focus is appropriate because we are interested in how computer systems are implemented to assist certain white-collar work functions rather than how secretaries or managers or professionals, as individuals, use such systems.

The 55 work groups in our sample can be classified by organizational mission as follows:

1. *Management and administration,* including personnel, contracts and grants, and financial management (Group I);
2. *Text-oriented professionals,* including public information, law, report writing and editing (Group II)
3. *Data-oriented professionals,* including economic modeling and forecasting, engineering design, and systems analysis (Group III)
4. *Secretarial, clerical, and technical support,* including reservations, payroll, order entry, and inventory control (Group IV)

The work groups ranged in size from 4 to 37; the average size was 10. While we cannot yet make causal inferences, it is interesting to note that in comparison with national data about white-collar work, these groups contain a substantially smaller proportion of support staff relative to professionals and managers.

Table I—Education level of employees

Education	%
Less than high school	0.2
High school	16.0
Some college	33.8
College degree	19.5
Masters or other equivalent degree	24.2
Ph.D., M.D. or equivalent degree	6.2

As Table I shows, the office employees in this sample are well educated. Almost a third have a graduate degree of some kind; only 16 percent have never attended college. These data suggest that implementing a computer system in an office with a work force of this kind should not be viewed as parallel to factory automation of a century ago, where poorly educated workers performed routine, low-level tasks.[†] Indeed, we found that the variety in work permitted by an office system was a significant predictor of its use and success.

Almost half these workers can keyboard quickly, and a sizable proportion have had some contact with a computer in a previous job. However, as Table II shows, background varies considerably by work group type. For example, none of the support group had regular prior experience with a computer while one-third of the data-oriented professionals had. The most proficient typists belong to the management/administration and to the text-oriented professional groups.

WHAT INFORMATION WORKERS DO

One of the research goals was to learn what kinds of tasks information workers perform. We wanted to know whether these tasks fall into clusters that could become focal points in implementing information systems. In addition, we wanted to identify any tasks, or clusters of tasks, that virtually everyone—from managers to clerks—performs.

†Some open-ended comments in the survey suggest, however, that computer systems are being implemented in some offices as if the office workers were only capable of performing routine tasks. Workers complain that training sessions explain only key functions rather than describing the system function. As a result, new users don't know enough about the system's real capabilities to exploit them efficiently or to solve problems when they arise.

Table II—Previous keyboard experience (percent of employees)

Previous Experience	Overall	Group I	Group II	Group III	Group IV
Previous work with computers					
None	53.4	61.3	51.1	42.4	75.0
Occasional	24.2	19.3	33.8	24.2	25.0
Regular	22.4	19.3	15.0	33.3	0.0
Proficiency in keyboarding or typing					
Not at all	0.9	1.0	0.0	1.0	0.0
A few fingers, slowly	8.5	9.9	6.7	11.1	0.0
A few fingers, quickly	16.6	12.4	12.6	24.2	0.0
Touch typing, slowly	25.5	19.0	23.0	32.3	50.0
Touch typing, quickly	48.5	57.9	57.8	31.3	50.0

Key to Groups: I=Management, administration
II=Text-oriented professionals
III=Data-oriented professionals
IV=Support staff

Table III—Information-related activities of office employees (percent who do each)

Information Activity Factors	Overall	Group I	Group II	Group III	Group IV
Activities: Factor 1 (Clerical work)					
Maintain files (.71)	56.9	58.5	48.5	55.6	64.3
Handle messages (.65)	48.5	53.7	36.8	37.4	62.9
Fill in forms (.62)	47.7	51.2	32.3	43.4	62.2
Process records (.59)	28.1	39.8	14.0	23.2	35.0
Keep activity logs (.55)	30.9	35.8	22.8	29.3	36.3
Maintain inventory (.55)	16.4	19.5	7.3	14.1	23.8
Keyboard text or data supplied by someone else (.53)	35.5	39.0	39.7	23.2	37.1
Administrative support (.37)	24.4	22.8	29.4	25.3	20.3
Activities: Factor 2 (Text manipulation)					
Write original material (.84)	65.9	67.5	77.9	77.8	44.8
Proofread and correct (.77)	63.1	61.8	77.9	60.6	51.7
Edit and rewrite (.85)	56.5	52.0	74.3	70.7	33.6
Activities: Factor 3 (Programming)					
Programming (.76)	20.2	14.6	18.4	47.5	7.7
Maintain a database (.70)	25.0	31.7	32.4	24.2	13.3
Statistical computation (.47)	27.1	35.0	36.0	26.3	12.6
Activities: Factor 4 (Numeric data manipulation)					
Fiscal operations (.78)	24.0	30.1	34.6	18.2	12.6
Distribute information (.61)	46.5	59.3	47.1	43.4	37.1
Statistical computation (.52)*	27.1	35.0	36.0	26.3	12.6
Develop forms (.45)	35.5	36.6	43.4	35.3	27.3
Administrative support (.38)*	24.4	22.8	29.4	25.3	20.3
Number of activities performed					
3 or fewer	20.0	14.6	13.2	18.2	32.2
4 to 6	30.3	27.6	34.6	31.3	28.0
7 to 10	29.1	35.0	33.8	24.2	23.0
11 or more	20.6	22.8	18.4	26.3	16.8
	100.0	100.0	100.0	100.0	100.0

*When a task loads on two different factors, the stronger loading is starred.
Key to Groups: I=Management, administration
II=Text-oriented professionals
III=Data-oriented professionals
IV=Support staff

We asked our respondents whether they performed each of 17 different activities; we chose these activities because they are thought to occur frequently in white-collar work and because they may be done with or without a terminal. Table III summarizes the results. Some tasks are performed by almost everyone; for example, the majority of workers in the sample at least occasionally write original material, proofread and correct, edit and rewrite, and maintain files. The tasks least widely performed are maintaining an inventory and programming.‡

To determine whether the 17 different tasks formed clusters, we used a factor analysis procedure. Factor analysis determines statistically whether some tasks are usually done in conjunction with others.§ This analysis generated four groups out of the 17 tasks (two tasks—administrative support and statistical computation—figure in two different groups). These task groups are shown in Table III along with the *factor loadings*. These numbers, given in parentheses after each of the tasks, show the strength of each item; the higher the factor loading, the more central that task is to the factor. For example, programming and maintaining a database are more central to Factor 3 than is statistical computation. Together the four factors account for about 60% of the variation in office work.

The factor analysis shows that white-collar employees tend to perform groups of tasks. For example, people who write original material also proofread and edit, but they tend not to program, maintain a database, or do statistical computation.

The four factors bear some resemblance to the four work group types. Factor 1 describes clerical and administrative activities and includes many tasks performed by the management/administration and by the support groups. These tasks primarily involve the management of text information. In contrast, Factor 2 involves creating and altering text, activities associated with the text-oriented professionals. Factor 3 suggests a more sophisticated knowledge of computers and reflects computer use by engineering and other applied-science professionals. Factor 4 involves the sophisticated manipulation of numeric data. Together Factors 3 and 4 reflect the activities of the data-oriented professionals.

Since the defined clusters of activities are at least superficially similar to the group types, it is not surprising to learn that the groups perform these tasks at different rates. For example, almost half of the data-oriented professionals do programming, in contrast to less than 8% of the support group employees. A less obvious example is processing records. Forty percent of the management group and 35% of the support group perform this task, but only 14% of the text-oriented professionals and 23% of the data-oriented professionals undertake records processing.

The lower portion of Table III shows how many different activities the members of each work group perform. More than 20% of the employees perform 11 or more of the 17 tasks and about an equal number perform 3 or fewer. Not surprisingly, management groups perform the widest range of tasks; those in support groups have the least diverse activities.

The figures reported in Table III should not be construed as measures of computer use. What they reflect is both the variety and the patterning in information handling activities among office workers. Successful implementation of information systems must reflect the patterning while accommodating the variety.

COMPUTER USE AND USER SATISFACTION

The task analysis just described suggests that a substantial number of the activities carried out by most white-collar workers could *potentially* be aided by a computer. In fact, 67% of the employees in this sample *do* interact directly with a computer during the regular course of their work, and another 26% expect to do so in the near future. Table IV breaks down the proportion of current users, expected users, and committed nonusers by specific occupational level in the organization. In contrast to widely publicized speculations about managerial and professional resistance, we found that employees at these occupational levels were both willing and able to convert to computerized systems; indeed, their responses to information technology are significantly more positive than those of support staff. It is noteworthy that, in offices where the technology is available, the overwhelming majority of employees in all occupational categories expect to be using computers as work tools in the near future.

Current users vary widely in the amount of time that they typically spend at a terminal. About half use the computer 30% or less of their working time; another fourth spend up to 70% of their time at a terminal; and the remaining fourth may use a computer up to full time. Forty percent of the users have their own terminal; the others use shared workstations. Very few (only 17%) have a printer near their desk.

The equipment in use represents many different models, makes, and vendors. Most sites have at least two different types of terminals and printers, and system architectures and configurations are quite diverse. Software choices are even more varied. Given this heterogeneity, we sought to discover whether there are generic characteristics of advanced office systems—summary dimensions—that would capture user satisfaction.

Table IV—Percentage of employees who use or will use computers at work

Occupational Category	Currently Use a Computer	Expect to Use a Computer	Do Not Expect to Use a Computer
Executive	36	46	18
Managerial	71	26	3
Professional	79	18	3
Technical	81	18	1
Secretarial	64	23	13
Clerical	73	21	6
Average	67	26	7

‡Groups for whom programming was a central function were not included in the research project.

§An orthogonal varimax rotation was employed, and only factors with an eigenvalue ≥ 1 were accepted.

Table V—Mean scores on satisfaction with features of the office computer system (4 = very satisfied, 1 = very dissatisfied)

Feature	Overall	Group I	Group II	Group III	Group IV
Features: Factor 1 (Functionality)					
Text or data alteration capability (.79)	3.4	3.4	3.5	3.4	3.2
Text or data entry capability (.75)	3.5	3.5	3.5	3.3	3.4
Organization of stored information (.74)	3.4	3.5	3.3	3.3	3.3
Information retrieval capability (.66)	3.3	3.4	3.4	3.1	3.2
Computer system's appropriateness for assisting your own particular job functions (.63)	3.3	3.5	3.4	3.2	3.4
Error detection and correction (.60)	3.2	3.2	3.0	2.9	3.4
Back-up to prevent accidental file loss (.55)*	3.1	3.4	3.0	3.0	3.0
Keyboard layout (.53)	3.5	3.6	3.6	3.2	3.6
Features: Factor 2 (Equipment performance)					
Promptness of maintenance (.82)	2.9	2.8	2.9	2.6	3.0
Quality of maintenance (.80)	3.0	3.1	2.9	2.8	3.2
Quality of printout (.67)	3.4	3.5	3.4	3.2	3.6
Quality of the video display (.53)	3.5	3.6	3.5	3.2	3.6
Back-up to prevent accidental file loss (.50)	3.1	3.4	3.0	3.0	3.0
Features: Factor 3 (Interaction)					
Quality of the operating manual (.78)	2.5	2.7	2.5	2.4	2.7
Type of dialog with the computer (.71)	3.3	3.4	3.3	3.0	3.3
Response time of the computer (.60)	2.8	2.9	2.9	2.3	2.9
Features: Factor 4 (Environment)					
Convenience and comfort of office furniture (.89)	3.1	3.1	3.0	2.6	3.2
Arrangement of equipment, furniture and space (.84)	2.8	2.9	2.8	2.5	3.0
Overall friendliness of the computer system	3.3	3.4	3.2	3.0	3.4

*When a task loads on two different factors, the stronger loading is starred.
Key to Groups: I=Management, administration
II=Text-oriented professionals
III=Data-oriented professionals
IV=Support staff

To investigate this question, we asked respondents to indicate their level of satisfaction with a number of very general features of their computer system, using 4-point rating scales. Table V lists these general characteristics and the average ratings for the entire sample and for each type of work group.

To learn whether computer system features could be organized to form major dimensions underlying user satisfaction, we again performed a factor analysis. That analysis generated the four factors under which system characteristics are listed in Table V. The factor loading appears in parentheses after each characteristic.

The factors can be interpreted as follows:

1. Functionality: how the system enters, alters, organizes, and stores information
2. Equipment performance, including speed and quality of maintenance
3. Interaction: whether the user has what is needed to interact effectively with the computer
4. Environment: adequacy, convenience, and comfort of equipment, furniture, and space.

Together, these four factors account for more than 60% of the variation in user satisfaction with computer system features.

Mean satisfaction scores in Table V show clearly that users are relatively happy with the functionality of their computer systems, especially with the text or data entry and editing capabilities. Reactions to equipment performance are mixed. The quality of printout and of video display get high marks, but promptness and quality of maintenance are given less satisfactory ratings. And, although users are unanimously unhappy about the environment in which they operate, user manuals get the worst ratings of any individual characteristic.

Within these general patterns, there are statistically significant differences between user groups. For example, support groups are relatively pleased with the error detection and correction capabilities of the system, but data-oriented professionals find the system unsatisfactory in this respect. Management groups are more pleased with backup features than are any other groups.[||]

In part, the differences in user judgments appear to be influenced by previous experience with computers. The data-oriented professionals are clearly the most dissatisfied with their computer systems; yet they are the heaviest users and

[||]In Table VI and following tables where mean scores are provided, differences on the order of .2 to .3 may generally be regarded as statistically significant.

Table VI—Mean scores on assessments of work performance (4 = very positive or satisfactory, 1 = very negative or unsatisfactory)

Computer Effects on Work and on Performance	Overall Mean	Group I Mean	Group II Mean	Group III Mean	Group IV Mean
On Work					
On speed of work	3.5	3.4	3.6	3.3	3.5
On quantity of work	3.4	3.3	3.6	3.4	3.4
On type of work	3.4	3.3	3.5	3.3	3.5
On quality of work	3.5	3.4	3.7	3.5	3.4
On quality of working life	3.4	3.2	3.4	3.3	3.4
On Performance					
Perceived productivity in own work	3.7	3.7	3.7	3.5	3.8
Perceived quality of own work	3.7	3.7	3.7	3.7	3.7

Key to Groups: I=Management, administration
II=Text-oriented professionals
III=Data-oriented professionals
IV=Support staff

more than a third of them regularly used computers in previous jobs. In contrast, the text-oriented professionals, more than half of whom had no prior computer experience, are the most satisfied with features of their office system.

We may surmise that the previous computer experience of the data-oriented workers makes them more critical and demanding of system performance. They are probably more knowledgeable about potential applications and compare their equipment with the more sophisticated and flexible systems now becoming available. Computers are not new work tools for them, and although they have high expectations of how systems should work, they are more likely than other employees to be using older, patchwork systems not really designed to handle interactive work.

On the other hand, the majority of text-oriented professionals are getting their first taste of the speed and convenience that computers can bring to their basic tasks—writing and editing text. The contrast with the tedious process of preparing material on the typewriter is indeed striking, and the computer provides a welcome increase in their ability to control production and meet deadlines. In addition, their briefer and more limited exposure to alternative computer systems tends to leave them contented with the one they have. This interpretation implies that organizations would do well to implement systems that can be upgraded as users become more experienced and as new options become available.

Since the four dimensions—functionality, equipment performance, interaction, and environment—seemed to summarize user satisfaction quite well, we wanted to see whether they would serve as general predictors of overall system friendliness, utilization levels, and satisfaction with the new technology.

We excluded overall friendliness, the last item listed in Table V, from the factor analysis because we regarded it as a product of all four system factors rather than as a specific feature. We used standard statistical regression procedures to test this hypothesis.# As we expected, all four factors were significantly related to ratings of overall system friendliness, but the association was statistically weakest for the environmental factor (Factor 4).

Next, we sought to determine whether satisfaction on the four summary dimensions could predict reported levels of computer use. Employing similar statistical procedures to test this relationship, we discovered that utilization is significantly associated only with functionality. That is, among the sets of features studied, user satisfaction with system functionality is what predicts the extent to which information technology is incorporated into regular work.

Finally, we investigated the degree to which satisfaction on these four dimensions could predict overall satisfaction with the new technology. For this outcome, we found two factors to be significant: functionality and environment.

These results support the following suggestions for implementing advanced information systems.

1. Successful implementation requires high ratings on all four computer system dimensions.
2. The implementation problems identified in the preceding discussion are not inherent computer problems—users are basically happy with system functionality. They are unhappy with system characteristics that *management can readily fix,* such as unhelpful technical manuals and unaccommodating office environments.

EFFECTS OF COMPUTER USE ON PERFORMANCE AT WORK

Ultimately, investigations of advanced information technology in private-sector contexts are expected to address the issue of how computer use affects work performance. A major incentive for investing in office computer systems, according to our sample respondents and to reports in other literature, is the expectation of increased productivity or decreased labor costs. In our survey we therefore asked respondents to estimate two sorts of outcomes:

1. The effects of computer use on the speed, quantity, type, and quality of white collar work of the sort done in

#We used standard linear regression techniques, with derived factor scores used as predictors of overall friendliness ratings, reported utilization levels, and satisfaction with the new technology.

Table VII—Mean scores on level of technology use and satisfaction

Use and Satisfaction	Overall Mean	Group I Mean	Group II Mean	Group III Mean	Group IV Mean
Extent of incorporation of new technology into daily work (3 = high level, 1 = low level)	2.2	2.0	2.2	2.3	2.3
Extent of satisfaction with new technology (4 = high level, 1 = low level)	3.2	3.1	3.4	3.1	3.1

Key to Groups: I=Management, administration
II=Text-oriented professionals
III=Data-oriented professionals
IV=Support staff

their office as well as its effects on the general quality of working life
2. The effects of computer use on the quality and the productivity of their own work

Table VI summarizes the average responses for the entire sample and for each type of work group. It is evident that most employees think computer use will have a favorable effect on various aspects of work performance. Mean estimates in the upper portion of the table range from 3.4 to 3.7 on a 4-point scale. However, as the bottom half of Table VI shows, respondents were even more positive about the effects of computer use on the quality and productivity of their own work. These are self-report results that need to be supplemented by archival data and other objective measures. We will pursue such analyses in later stages of this research project. However, it is interesting to note that respondents were significantly more positive about how the computer might affect the quantity and quality of their own work than they were about how it might affect the quality of working life.

Between-group differences in judgments about work effects shown in Table VI mirror previously reported differences in computer system satisfaction. The data-oriented professionals are less positive than other groups, while the text-oriented professionals are the most enthusiastic. Not surprisingly, support groups report the greatest positive effect of computer use on productivity. Their estimate is consistent with the conclusions of others (e.g., Johnson and Taylor,[2]) that productivity gains through computerization of office work are most rapidly and easily achieved in secretarial and other support tasks.

We have described how satisfaction on the four summary dimensions of computer systems predicts the level of utilization and overall satisfaction with the new work tool. It is appropriate to end this preliminary discussion of research results by asking whether level of utilization and degree of satisfaction are directly linked to assessment of productivity. For the time being, the answer is No. Table VII shows how work groups rated the extent to which they had incorporated the new technology into their daily work and their satisfaction with it. All groups report only intermediate levels of utilization. Moreover, statistical tests show no significant relationship between levels of use and levels of satisfaction with the technology. Nor is there a significant relationship between either of these variables and the very high estimate (3.7) of the computer's effect on worker productivity. In one sense, the lack of a connection is not surprising; decades of organizational research have been unable to demonstrate a consistent relationship between satisfaction and productivity at work (see, e.g. Brayfield and Crockett[1]). On the other hand, given the billion-dollar cost problems associated with white-collar absenteeism, satisfaction with work would appear to be a necessary although not sufficient condition for productivity.

ORGANIZATIONAL CHARACTERISTICS THAT AFFECT IMPLEMENTATION

In order to link our characterization of activity in computerized offices with other national studies of the quality of working life, we included in our survey some standard measures of organizational characteristics.** These measures tapped such aspects of organizations as centralization of decision making, employee autonomy, variety in work, challenge in work, and the organization's orientation toward change. Our hypothesis was that the organizational context for substantial change, such as the introduction of innovative office technology, would certainly affect implementation success.

Survey results showed that two features of organizations do predict how completely users integrate information systems into their daily work and how happy they are with them.

1. *Variety in work* significantly predicts the level of system use. This finding, coupled with the relatively high education level of the workers who will be using office computers, reinforces our concern that implementing information systems in the white-collar environment not be patterned on automating routine factory work.
2. An organization's *orientation toward change* significantly predicts both level of system use and user satisfaction with the new technology. White-collar workers appear to adapt more readily to innovative information systems when they work for organizations where such change is viewed as a positive, problem-solving, and achievable goal.

**These measures were developed by the University of Michigan Survey Research Center and have been widely used in organizational research (e.g., Quinn and Shepard[3]).

Because an organization's change orientation plays such an important role in implementing an information system, we will explore it more thoroughly in subsequent research. Interview data and other descriptive information will help in determining the ingredieints of positive change orientation and suggest how organizations can best manage the process of embedding technology in information-related work.

These organizational findings, coupled with analyses of user satisfaction and dissatisfaction described above, strengthen our belief that problems with implementing computer systems in white-collar settings are not technological in nature. Users think the technology itself is fine. What affects how much and how well it is used are organizational, environmental, and training matters that organizations should be able to address.

CONCLUSIONS

White-collar work can be described in terms of four systematic clusters of information-handling activities. They are carried out by well-educated employees who, across sex and occupational category, have some keyboarding skills. Many have used a computer in a previous job.

Contrary to popular reports, senior managers and professionals are not resistant to using computers in organizations where advanced office technology has been installed. Indeed, more than two-thirds of them already use a computer, and almost all of the rest expect to do so in the near future.

For the most part, workers give the new technology per se high marks. They are, for the most part, satisfied with the functionality of the applications available to them. However, they are distinctly unhappy with some features that affect the user interface—for example, user manuals and promptness of maintenance. And they find the environment in which the system is configured uncomfortable or inappropriate.

Four basic dimensions were found to summarize user reactions to the new systems and to predict both how much the system is used and how well users like it. Among them, functionality features predict how thoroughly the system is incorporated into daily work, and together with environmental characteristics they predict overall satisfaction with the new technology.

Surveyed employees estimate that the computer will have very positive effects on the quality and productivity of their own work. However, they are less enthusiastic in their assessment of how it will affect the quality of working life.

Organizational characteristics appear to be as important as computer characteristics in the successful implementation of an innovative office system. In particular, variety of work and the organization's orientation toward change are significant predictors of how much the new system will be used and how satisfied users will be.

These findings suggest that the most critical problems in implementing information systems are not technological ones. Instead, they involve basic characteristics of the organization: how its structures work, how it responds to change, and how adequately it considers employee needs in designing the user-computer interface.

REFERENCES

1. Brayfield, A., and W. Crockett. "Employee Attitudes and Employee Performance." *Psychological Bulletin,* 52 (1955), pp. 415–422.
2. Johnson, B., and J. Taylor. "Innovation in Word Processing," Interim Report to the National Science Foundation, Grant No. ISI-8110791, 1982.
3. Quinn, R., and L. Shepard. *The Quality of Employment Survey.* Ann Arbor: Institute for Social Research, 1974.

An Interactive Display Environment, or knitting sheep's clothing for a wolf

by ROBERT P. O'HARA
The IBM Corporation
Yorktown Heights, New York

ABSTRACT

AIDE, An Interactive Display Environment, attempts to improve the usability of a traditional timesharing system by exploiting the use of a powerful personal computer with a large, high-resolution all-points-addressable (APA) display as a terminal. No changes to the timesharing system are required, and existing application programs dependent on alphanumeric display terminals run unchanged. A sample terminal session illustrates AIDE in use. Ways in which the timesharing system might be modified to further exploit the display are suggested.

MOTIVATION

The display interface of an interactive system represents a major part of the face the system presents to the user. Most traditional timesharing systems such as VM/370 CMS[4,11] (VM/CMS) and UNIX[10] were designed with typewriter terminals as the assumed user terminal. In such systems the user types commands as strings of words; the system types back a response that may be many lines long. This conversation is recorded on the typewriter terminal's paper. If the user wishes to review an earlier portion of the conversation, it is readily available.

Display terminals (hereafter called displays) have supplanted typewriter terminals as the standard interactive terminal. They offer advantages such as very high speed operation and the ability to present a formatted display. Programs that use these formatted *fullscreen* displays, such as visual editors and menu-driven applications, offer increased ease of use and productivity over the previous typewriter-mode programs, but typically fit poorly with the rest of the system. The user switches between typewriter mode and fullscreen mode during the interactive session, losing whatever was on the screen during a mode switch. The typewriter-mode interface at a display is essentially unchanged from that at a typewriter terminal. The conversation that previously was recorded on the typewriter paper now scrolls off the screen and is lost, as if someone came along and tore off the paper every 24 lines. In other words, a valuable function (that of review of previous commands and responses) has been lost.

Several window/terminal management systems that provide more than "typewriter mode on a display" have been implemented. Notable for support of alphanumeric displays are the IBM TSO Session Manager,[7] the RIG Virtual Terminal Management System at the University of Rochester,[5] BRUWIN at Brown University,[8] the Automated Desk,[14] and Hartman's TOY system.[3]

I believe that large-screen all-points-addressable (APA) displays are as important an advance in function and usability as alphanumeric displays were over typewriter terminals. Far more information can be displayed on them, and their APA capability allows the use of multiple fonts, graphics, and images. Systems such as Smalltalk[12] and the Xerox Star[6] have been designed with APA displays in mind. These systems are visually oriented, designed around a pointing device such as a mouse. They do not offer either a traditional command language interface or a "fill in the blanks" menu interface, but rather they define new paradigms for the user-system interface, based on the capabilities of an APA display and a powerful personal computer.

In this project I chose to explore how one can use such displays with an existing interactive system. It is often not practical to rewrite the system to support an APA display. Existing programs, many with dependencies on alphanumeric displays, must continue to operate.

GOALS

The goals of the project were to

- enhance the usability and function of the system by fully exploiting the APA display.
- provide a base for an evolution to new modes of display usage and user interfaces.
- allow all existing programs of the interactive system to run unchanged, including "fullscreen" programs.
- not modify the existing interactive system.

GUIDING PRINCIPLES

So how does one go about designing a better user interface? The approach used in developing AIDE was to develop a list of principles to guide the design. The intent was to have these principles embody good human factors, and thus assist in achieving that better interface. The design principles were the following:

- Preserve the context of the user's conversation with the system. This is important because the relationships among commands entered during a session are often as important as the commands themselves; there is continuity to the user/system conversation.

  Show more of it. The typical display terminal has 24 lines of 80 characters. Note that this is not even one page of text. When the screen is split into two or more windows, very little information can be displayed in each window. It's difficult to be productive when one views the world through a keyhole. Large-screen displays such as the one used by AIDE can display over 90 rows of 100 characters, and even this is not too big!

  Avoid its loss. Most systems clear the screen when entering and leaving the editor. It is not uncommon to observe users copying information from the screen with pencil and paper before it is lost. AIDE creates a separate window for the editor or other fullscreen programs, preserving the information displayed in the typewriter-mode window.

  Avoid its disruption. Asynchronous messages from other users and programs are typically displayed immediately on the screen, interspersed with and interrupting whatever else is there. If the user is within an editor, the editing session is typically interrupted, the screen cleared, and the message displayed. Before resuming the editing session, the user must copy the message on paper if it is not to be lost. AIDE places asynchronous messages in a separate window to avoid disrupting the current context.

- Provide feedback to user actions. Norman emphasizes the importance of feedback in preventing many classes of user errors.[9] AIDE provides instant feedback within pop-up menus by using reverse video to highlight the command pointed at by the user. Smooth scrolling of the message window one raster line at a time gives the user instant feedback and easy control of the amount of scrolling desired. It seems to eliminate the problem of windowing versus scrolling common on many systems.[1]
- Don't hide the state of the system. Norman also observes that "mode errors occur when the person believes the system is in one state (mode), whereas it is actually in another." AIDE indicates the modes of the system in several different ways:
  1. Different cursor shapes are used to indicate the different states of the system and terminal:

     Insert/replace mode of the character cursor. AIDE changes the shape of the character cursor itself, from an underscore (replace mode) to a reverse-video box (insert mode).

     Typewriter/fullscreen mode. The pointing cursor on the screen changes shape when fullscreen mode is entered, indicating that it may now be used to move the character cursor within the fullscreen window.

     Screen manipulation menus. A different pointing cursor appears when the screen manipulation menus are active. This informs the user that the pointing cursor is now to be used to make a selection from the menu.
  2. Separate windows are used for alphanumeric fullscreen programs. Displaying the fullscreen editor in a separate window superimposed on the typewriter-mode window clearly indicates that the user has entered a different environment. Burying the fullscreen window upon leaving the editor clearly shows the return to the system level.
  3. Recursion levels in the system, such as within the editor, are shown by surfacing additional windows, with the current level of recursion displayed in the topmost window. With a minor change to the editor an additional fullscreen window is created when additional files are edited. This has been a very useful approach with VM/CMS, for many of the most used programs such as electronic-mail sending and reading, file-list manipulations, system help information and so on are based on the editor.
- Eliminate modes where possible. The typewriter mode of VM/CMS at a display terminal presents six different states to the user. The rules governing which key to press when are very complex. AIDE simplifies life at the terminal by handling typewriter mode for the user. For example, the CLEAR key is pressed when needed.

IMPLEMENTATION OVERVIEW

Connecting non-IBM equipment to IBM computers is often a challenge; connecting the personal computer on which AIDE is implemented to the IBM host system proved no exception.

A Three Rivers Perq computer is the terminal used by AIDE. The Perq has a specially microcoded processor that executes PASCAL intermediate code at up to 1 million instructions per second and a 24-megabyte hard disk. A 768 × 1024 100 points-per-inch black and white APA display directly displays 96k of the 1 megabyte of memory. A *Rasterop* instruction copies any size area of memory, including the memory displayed on the screen, to any other area of memory in one refresh cycle of the display. This allows for very fast screen operations. The Perq also has a keyboard and tablet with a pointing device. The pointing device is a Summagraphics Bit Pad tablet and puck. The Perq connects to an IBM Series/1 computer over a 9600 baud full-duplex asynchronous line. The Series/1 connects to an IBM S/370 model 3031 AP computer through a block-multiplex channel.

Software

AIDE is written in PASCAL and runs on the Perq. It uses a window management system called JAWS (Jaws A Window System)[2] that is also written in PASCAL. In fact, all of the software running on the Perq is written in PASCAL, including the operating system, since the Perq is microcoded to efficiently execute PASCAL intermediate code.

AIDE makes the Perq appear as an IBM 3101 ascii teletype display terminal, which is one of the terminals supported by the IBM Series/1 Yale Ascii Terminal Communications System Installed User Program (the Yale IUP, program number 5796-RBT) running on the Series/1. The Yale IUP in turn makes the IBM 3101 look like an IBM 3277 display terminal to the System/370 model 3031 computer. The 3031 is running the VM/CMS operating system, and it is to CMS that the user finally logs on.

Operation

The result of all this deception is that CMS believes the user is at a locally attached IBM 3277 alphanumeric display terminal. AIDE views the terminal screen as a 24 by 80 character array. When the terminal is in typewriter mode (line at a time operation) AIDE interprets this screen and displays user input, system output, and messages in various windows, as illustrated in the sample terminal session below. When a fullscreen program is invoked, AIDE recognizes this and creates a 24 by 80 screen and program function (PF) key pad.

AIDE also has a small repertoire of actions that can be invoked from programs running on the host computer. These actions include creating and hiding fullscreen and line-mode windows.

SAMPLE TERMINAL SESSION

This sample terminal session illustrates the facilities provided by AIDE. It is a series of snapshots of the display screen.

Unfortunately the printer used does not copy the right half-inch of the display screen, so some clipping of the images has occurred.

Figure 1 shows the terminal screen shortly after the user has logged on to VM/CMS. There are three windows visible on a restful gray background. In the upper right-hand corner is a window that displays the current date and time. The largest window is titled "VM/CMS Console Log." It displays user input and system output of the session while in typewriter mode. It is analogous to the roll of paper in a typewriter terminal. When the window fills with data, the data are scrolled up to allow new lines to be displayed. The window at the bottom of the screen is titled "VM/CMS Console Input." The character cursor is displayed in this window; here the user enters commands and other input. In the lower right-hand corner of this window a status message is displayed. This status message is generated by VM/CMS, not by AIDE. The console input window is simply viewing the bottom two lines of the simulated display terminal, which is the command entry area for VM/CMS.

Near the center of the screen is a small arrow pointing to the upper left. This is the *pointing cursor* that follows the position of the pointing device on the Perq's tablet. Its image on the screen is maintained by special hardware in the Perq.

The other user has responded to the initial message of "howdy" by sending several messages of his own. These are displayed in a separate window titled "External Messages." This window surfaces automatically whenever a new message is received from another user or system. VM/CMS prefixes each message with a timestamp and its origin.

Since this window overlies the console window, the user may wish to *bury* it beneath the console window. To do this the pointing device is used to move the cursor (pictured as an arrow on the screen) so that it points at the message window. Then one of the buttons on the pointing device is pressed.

Instantly, a *pop-up menu* appears (Figure 2) at the position of the cursor. The cursor's shape is changed to signal that a selection is to be made from the menu. As the cursor is moved over the menu, the commands are displayed in reverse video, providing the user with instant feedback as to what command will be issued. Since the user wishes to bury the message window, Bury is selected by pressing the pointing device. The menu disappears and the window is buried beneath all other windows. Its location on the screen has not changed. Had the user chosen instead not to select any of the commands, pressing the pointing device while the cursor is outside of the menu would remove it from the screen without executing any commands.

In Figure 3 the user has issued the rl command (a synonym for RDRLIST) which lists the files in the user's mailbox (virtual card reader). This command uses the system editor to edit the list of files in a fullscreen manner. AIDE recognizes that a fullscreen program has been invoked and in response it creates a window the size of the simulated terminal display screen (24 by 80 characters) and a program function key pad with the 12 PF keys arranged as they are on an IBM 3277 terminal. The typewriter-mode input window has disappeared because it is not useful while in the editor. The cursor's shape has changed to indicate that a fullscreen program is active.

The pointing device can be used to directly position the character cursor on the simulated 3277 screen, avoiding the use of the traditional cursor movement keys. Positioning the character cursor in this manner is much faster than holding a key and waiting for the cursor to crawl across the screen. However, for small cursor movements of a couple of characters or so the cursor movement keys on the Perq keyboard are faster than the pointing device. This is especially true if other control keys such as insert and delete are being used.

The same pointing device is used to press the PF keys displayed on the screen: the pointing cursor is moved to the PF key and pressed. Thus PF key oriented programs like RDRLIST become very easy to use. To view a file in the list, the user points at the file name and presses to move the character cursor, then points at PF 11 and presses to press the PF key.

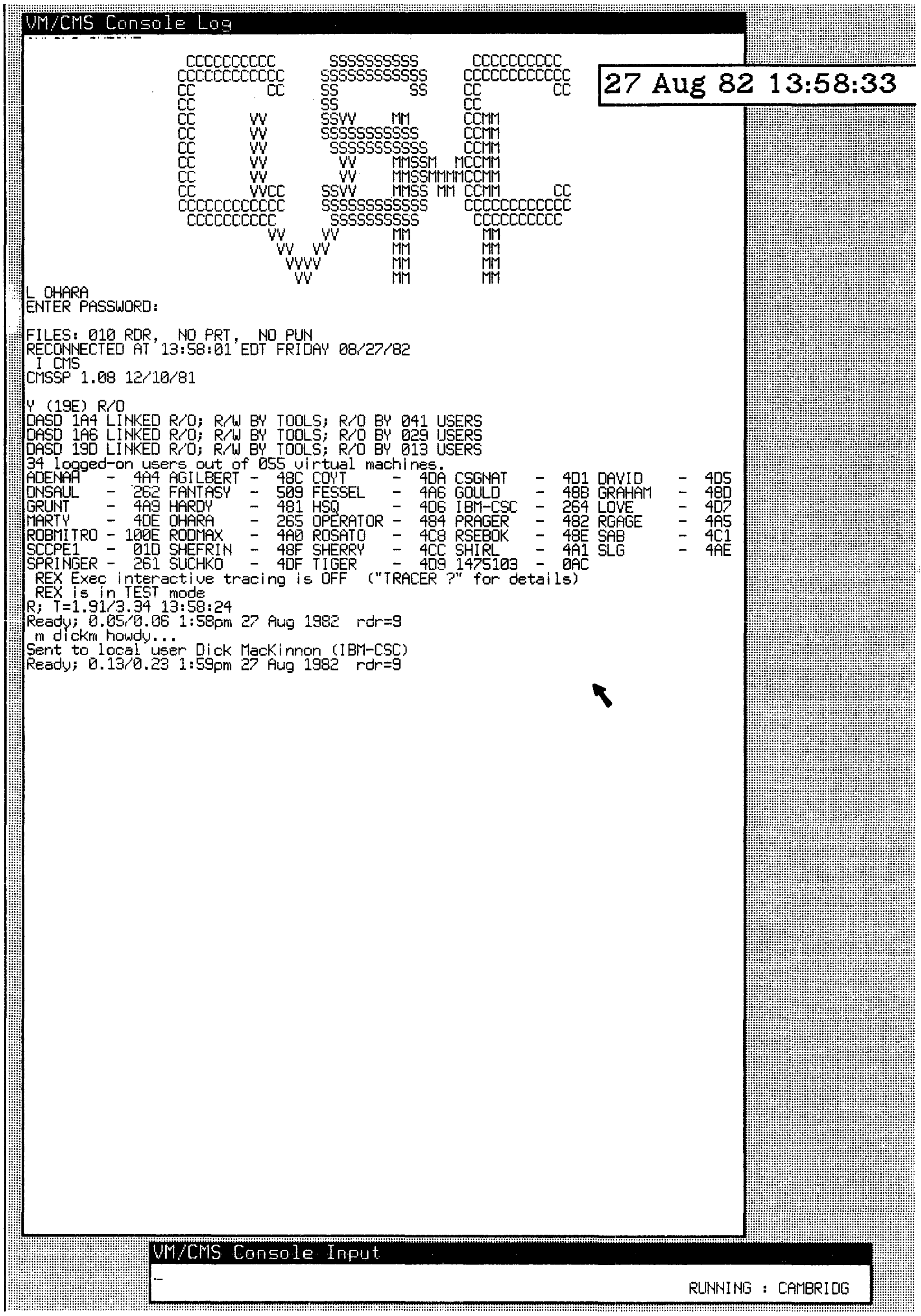
The PEEK command thus invoked calls the editor recursively. With a small change to the profile of the PEEK command AIDE is requested to create another fullscreen window, visualizing the recursive call to the user. The first fullscreen window is now inactive, because the RDRLIST program is now dormant until the PEEK command returns to it. The user has decided to reply to the note displayed by PEEK. From the PEEK screen the NOTE command is issued (Figure 4), which invokes the editor in a recursive manner. AIDE again creates another fullscreen window.

Looking at Figure 4, the reader might conclude that the display screen appears rather cluttered with windows, and that AIDE has made the terminal environment more confusing than before. For the user at the terminal, this does not appear as a problem. The windows appear in direct response to user actions, under control of the user, over time as the session proceeds. An analogy to working at a desk can be drawn. During the course of the day, a person places papers and opens books on the desk. Someone else viewing the desk might see a clutter, but to the person at the desk it is manageable and useful to have many pieces of information partially visible at once.

It is important to contrast the screen image shown here with that of a standard display terminal. With AIDE, the user sees exactly where he or she is in the interactive session. At a standard terminal the user would view only the current screen image, and the contents of the screen prior to entering the fullscreen program, including all external messages, would have been lost.

Later in the session the user displays the file sent from the colleague he or she earlier conversed with. Wishing to review that conversation, the user points at the buried message window and presses the pointing device. The menu pops up and the user points at the Surface command.

Pressing the pointing device again causes the window to surface instantly (Figure 5). To scroll back to view the previous messages, the user points to the scroll arrow on the right side of the window. Pressing the pointing device causes the window to scroll smoothly one raster line at a time over the data. The speed of scrolling is related to where on the scroll arrow the user points. The scroll arrow is located on the right-hand side of the message window. Close to the center the rate is slow; at the heads it is quite fast. The smooth



VM/CMS Console Log

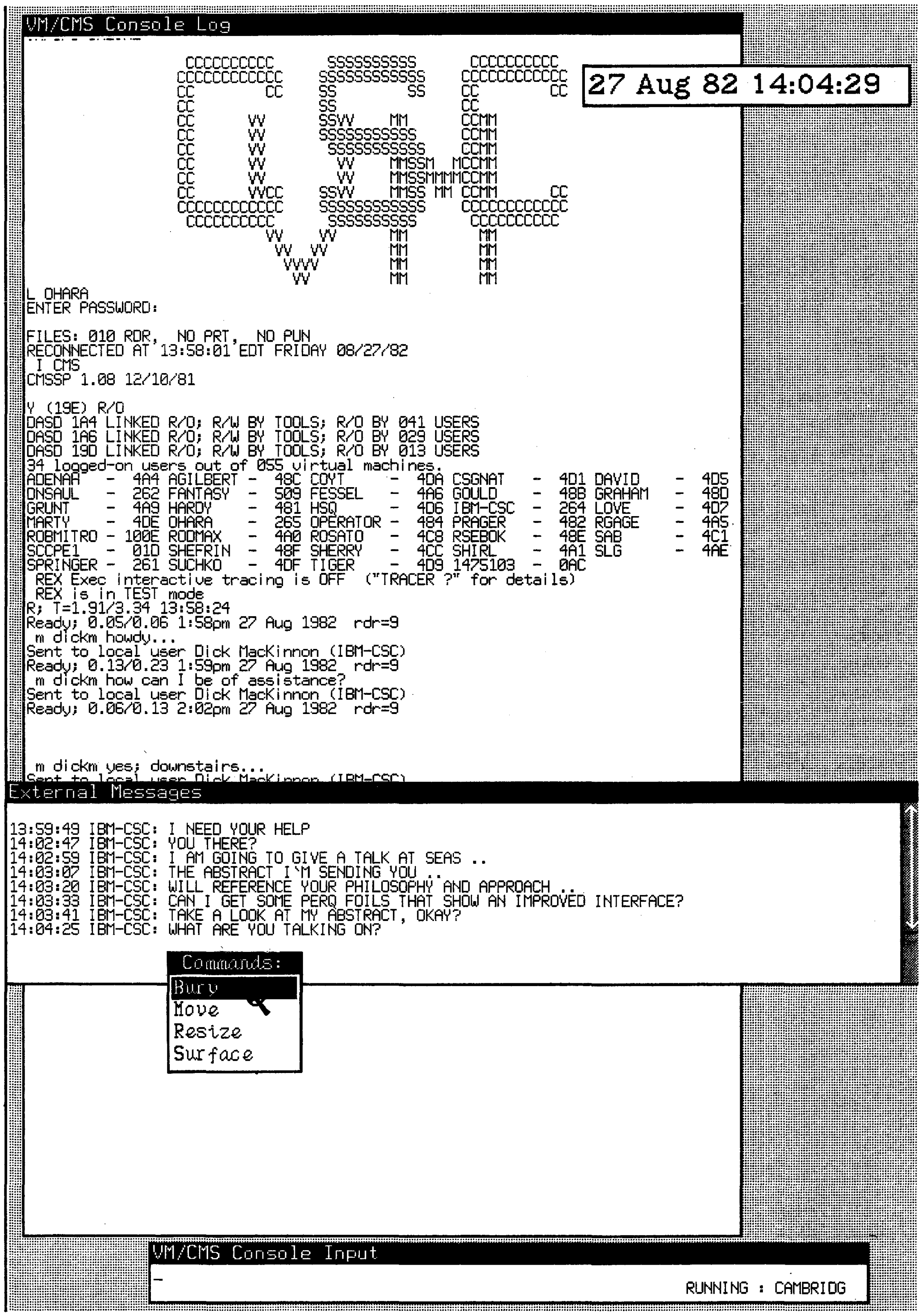
27 Aug 82 13:58:33

```
                  CCCCCCCCCC      SSSSSSSSSS      CCCCCCCCCC
                 CCCCCCCCCCCC    SSSSSSSSSSSSS   CCCCCCCCCCCC
                 CC        CC    SS         SS   CC        CC
                 CC              SS              CC
                 CC      VV      SSVV    MM      CCMM
                 CC      VV      SSSSSSSSSSS     CCMM
                 CC      VV       SSSSSSSSSSS    CCMM
                 CC      VV        VV    MMSSM  MCCMM
                 CC      VV        VV    MMSSMMMMCCMM
                 CC      VVCC    SSVV    MMSS MM CCMM      CC
                 CCCCCCCCCCCC    SSSSSSSSSSSSS   CCCCCCCCCCCC
                  CCCCCCCCCC      SSSSSSSSSS      CCCCCCCCCC
                           VV    VV      MM        MM
                            VV  VV       MM        MM
                             VVVV        MM        MM
                              VV         MM        MM
L OHARA
ENTER PASSWORD:

FILES: 010 RDR,   NO PRT,   NO PUN
RECONNECTED AT 13:58:01 EDT FRIDAY 08/27/82
 I CMS
CMSSP 1.08 12/10/81

Y (19E) R/O
DASD 1A4 LINKED R/O; R/W BY TOOLS; R/O BY 041 USERS
DASD 1A6 LINKED R/O; R/W BY TOOLS; R/O BY 029 USERS
DASD 19D LINKED R/O; R/W BY TOOLS; R/O BY 013 USERS
34 logged-on users out of 055 virtual machines.
ADENAA   -  4A4 AGILBERT -  48C COYT     -  4DA CSGNAT   -  4D1 DAVID    -  4D5
DNSAUL   -  262 FANTASY  -  509 FESSEL   -  4A6 GOULD    -  48B GRAHAM   -  48D
GRUNT    -  4A9 HARDY    -  481 HSQ      -  406 IBM-CSC  -  264 LOVE     -  4D7
MARTY    -  4DE OHARA    -  265 OPERATOR -  484 PRAGER   -  482 RGAGE    -  4A5
ROBMITRO - 100E RODMAX   -  4A0 ROSATO   -  4C8 RSEBOK   -  48E SAB      -  4C1
SCCPE1   -  01D SHEFRIN  -  48F SHERRY   -  4CC SHIRL    -  4A1 SLG      -  4AE
SPRINGER -  261 SUCHKO   -  4DF TIGER    -  4D9 1475103  -  0AC
 REX Exec interactive tracing is OFF  ("TRACER ?" for details)
 REX is in TEST mode
R; T=1.91/3.34 13:58:24
Ready; 0.05/0.06 1:58pm 27 Aug 1982  rdr=9
 m dickm howdy...
Sent to local user Dick MacKinnon (IBM-CSC)
Ready; 0.13/0.23 1:59pm 27 Aug 1982  rdr=9
```

VM/CMS Console Input

```
_
                                                RUNNING : CAMBRIDG
```

Figure 1—Shortly after logon to VM/CMS

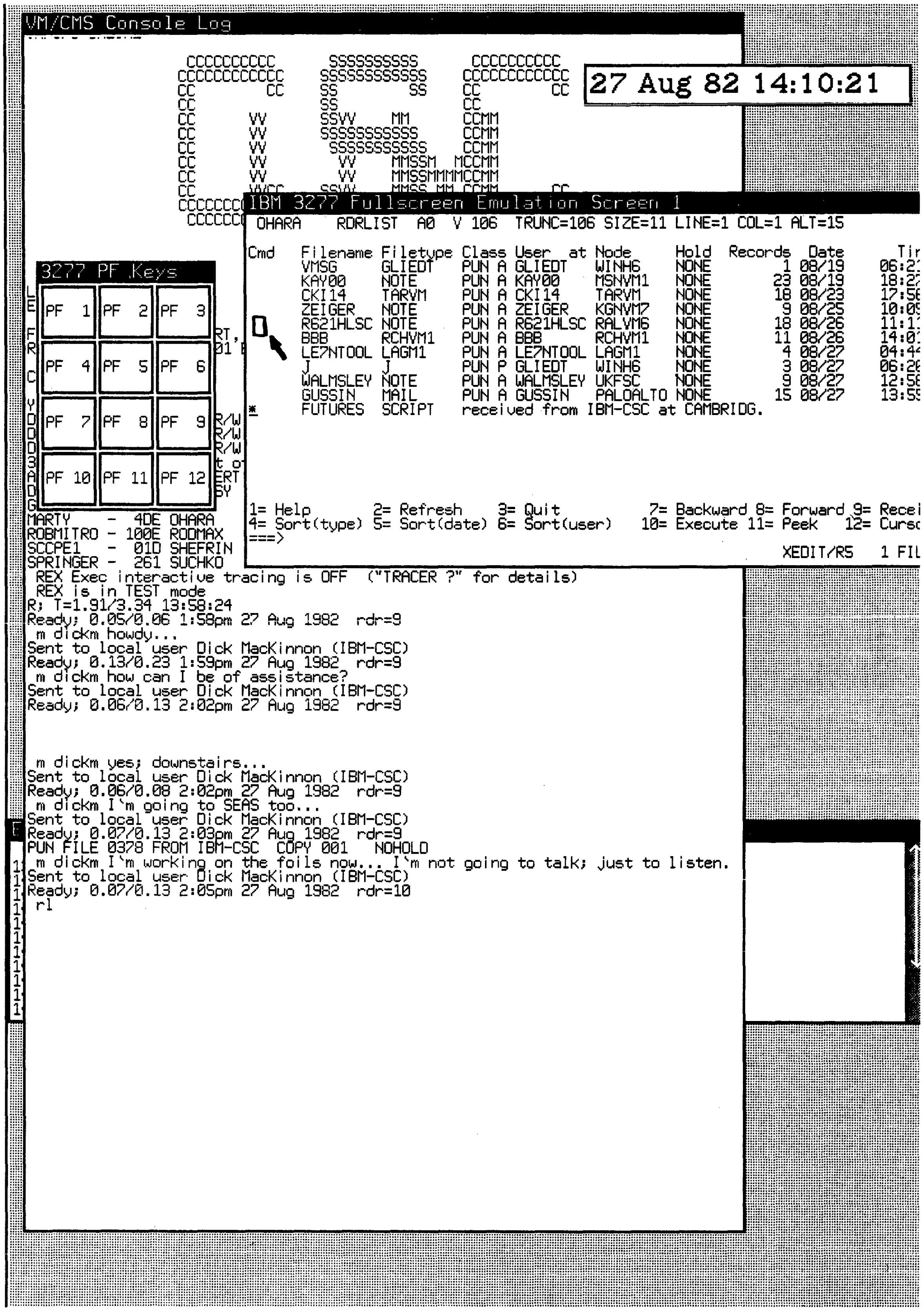


```
 CCCCCCCCCC      SSSSSSSSSS      CCCCCCCCCC
CCCCCCCCCCCC    SSSSSSSSSSSS    CCCCCCCCCCCC
CC        CC    SS        SS    CC        CC
CC              SS              CC
CC      VV      SSVV    MM      CCMM
CC      VV      SSSSSSSSSSS     CCMM
CC      VV       SSSSSSSSSSS    CCMM
CC      VV        VV    MMSSM  MCCMM
CC      VV        VV    MMSSMMMMCCMM
CC      VVCC    SSVV    MMSS MM CCMM      CC
CCCCCCCCCCCC    SSSSSSSSSSSS    CCCCCCCCCCCC
 CCCCCCCCCC      SSSSSSSSSS      CCCCCCCCCC
          VV    VV      MM        MM
           VV  VV       MM        MM
            VVVV        MM        MM
             VV         MM        MM
L OHARA
ENTER PASSWORD:

FILES: 010 RDR,  NO PRT,  NO PUN
RECONNECTED AT 13:58:01 EDT FRIDAY 08/27/82
 I CMS
CMSSP 1.08 12/10/81

Y (19E) R/O
DASD 1A4 LINKED R/O; R/W BY TOOLS; R/O BY 041 USERS
DASD 1A6 LINKED R/O; R/W BY TOOLS; R/O BY 029 USERS
DASD 19D LINKED R/O; R/W BY TOOLS; R/O BY 013 USERS
34 logged-on users out of 055 virtual machines.
ADENAR   -  4A4 AGILBERT -  48C COYT     -  4DA CSGNAT   -  4D1 DAVID    -  4D5
DNSAUL   -  262 FANTASY  -  509 FESSEL   -  4A6 GOULD    -  48B GRAHAM   -  48D
GRUNT    -  4A9 HARDY    -  481 HSQ      -  4D6 IBM-CSC  -  264 LOVE     -  4D7
MARTY    -  4DE OHARA    -  265 OPERATOR -  484 PRAGER   -  482 RGAGE    -  4A5
ROBMITRO - 100E RODMAX   -  4A0 ROSATO   -  4C8 RSEBOK   -  48E SAB      -  4C1
SCCPE1   -  01D SHEFRIN  -  48F SHERRY   -  4CC SHIRL    -  4A1 SLG      -  4AE
SPRINGER -  261 SUCHKO   -  4DF TIGER    -  4D9 1475103  -  0AC
 REX Exec interactive tracing is OFF  ("TRACER ?" for details)
 REX is in TEST mode
R; T=1.91/3.34 13:58:24
Ready; 0.05/0.06 1:58pm 27 Aug 1982  rdr=9
 m dickm howdy...
Sent to local user Dick MacKinnon (IBM-CSC)
Ready; 0.13/0.23 1:59pm 27 Aug 1982  rdr=9
 m dickm how can I be of assistance?
Sent to local user Dick MacKinnon (IBM-CSC)
Ready; 0.06/0.13 2:02pm 27 Aug 1982  rdr=9



 m dickm yes; downstairs...
Sent to local user Dick MacKinnon (IBM-CSC)
```

```
13:59:49 IBM-CSC: I NEED YOUR HELP
14:02:47 IBM-CSC: YOU THERE?
14:02:59 IBM-CSC: I AM GOING TO GIVE A TALK AT SEAS ..
14:03:07 IBM-CSC: THE ABSTRACT I`M SENDING YOU ..
14:03:20 IBM-CSC: WILL REFERENCE YOUR PHILOSOPHY AND APPROACH ..
14:03:33 IBM-CSC: CAN I GET SOME PERQ FOILS THAT SHOW AN IMPROVED INTERFACE?
14:03:41 IBM-CSC: TAKE A LOOK AT MY ABSTRACT, OKAY?
14:04:25 IBM-CSC: WHAT ARE YOU TALKING ON?
```

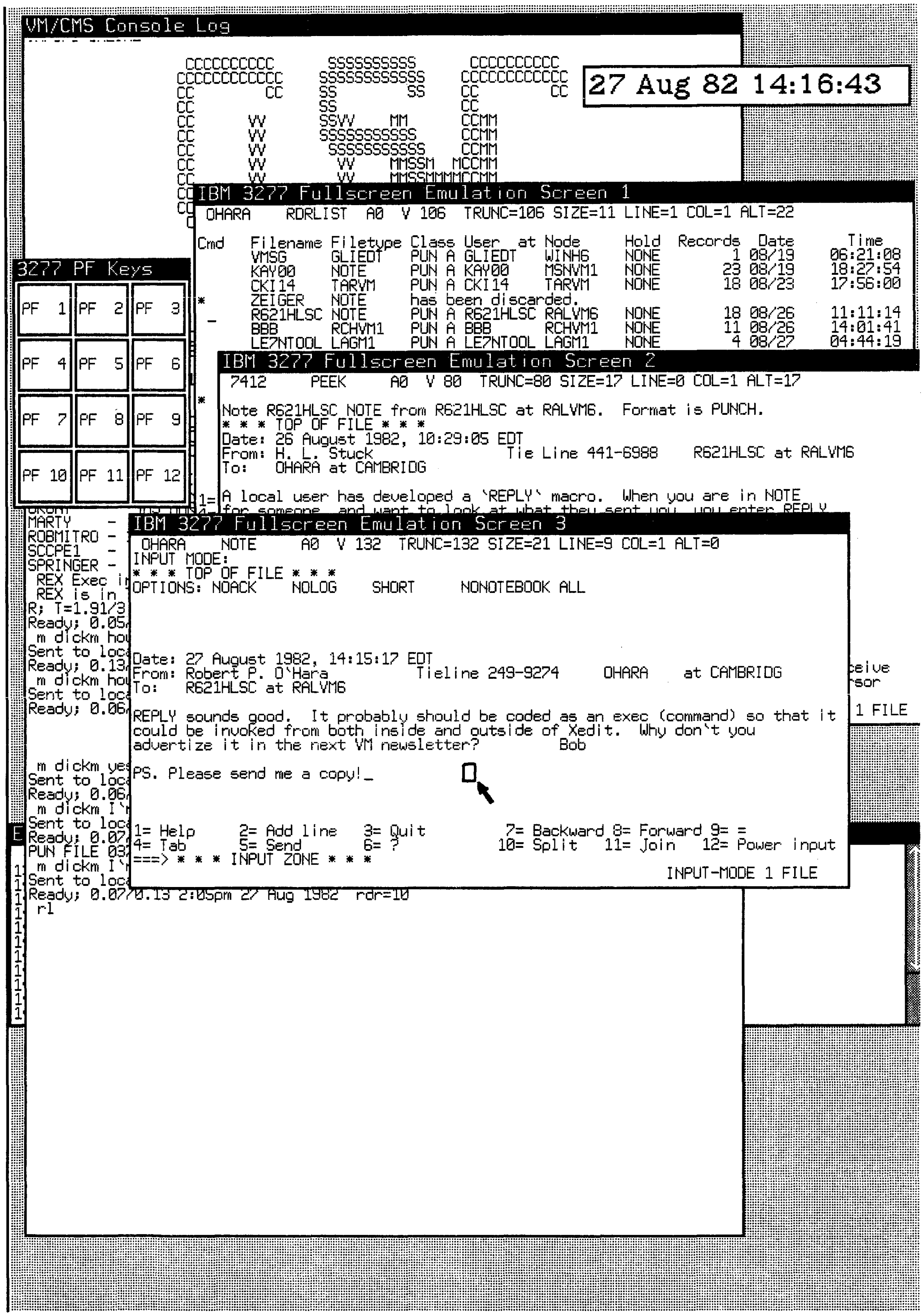
```
_
```

Figure 2—The message window and pop-up menu



```
VM/CMS Console Log

27 Aug 82 14:10:21

IBM 3277 Fullscreen Emulation Screen 1
 OHARA    RDRLIST  A0  V 106  TRUNC=106 SIZE=11 LINE=1 COL=1 ALT=15

Cmd    Filename Filetype Class User     at Node     Hold  Records  Date      Ti
       VMSG     GLIEDT   PUN A GLIEDT      WINH6    NONE        1 08/19     06:2
       KAY00    NOTE     PUN A KAY00       MSNVM1   NONE       23 08/19     18:2
       CKI14    TARVM    PUN A CKI14       TARVM    NONE       18 08/23     17:5
       ZEIGER   NOTE     PUN A ZEIGER      KGNVM7   NONE        9 08/25     10:0
       R621HLSC NOTE     PUN A R621HLSC    RALVM6   NONE       18 08/26     11:1
       BBB      RCHVM1   PUN A BBB         RCHVM1   NONE       11 08/26     14:0
       LE7NTOOL LAGM1    PUN A LE7NTOOL    LAGM1    NONE        4 08/27     04:4
       J        J        PUN P GLIEDT      WINH6    NONE        3 08/27     06:2
       WALMSLEY NOTE     PUN A WALMSLEY    UKFSC    NONE        9 08/27     12:5
       GUSSIN   MAIL     PUN A GUSSIN      PALOALTO NONE       15 08/27     13:5
*      FUTURES  SCRIPT   received from IBM-CSC at CAMBRIDG.

1= Help        2= Refresh    3= Quit           7= Backward 8= Forward 9= Recei
4= Sort(type)  5= Sort(date) 6= Sort(user)    10= Execute 11= Peek    12= Cursc
===>
                                                          XEDIT/R5    1 FIL

3277 PF Keys
PF 1   PF 2   PF 3
PF 4   PF 5   PF 6
PF 7   PF 8   PF 9
PF 10  PF 11  PF 12

MARTY     -  4DE OHARA
ROBMITRO  - 100E RODMAX
SCCPE1    -  01D SHEFRIN
SPRINGER  -  261 SUCHKO
 REX Exec interactive tracing is OFF  ("TRACER ?" for details)
 REX is in TEST mode
R; T=1.91/3.34 13:58:24
Ready; 0.05/0.06 1:58pm 27 Aug 1982  rdr=9
 m dickm howdy...
Sent to local user Dick MacKinnon (IBM-CSC)
Ready; 0.13/0.23 1:59pm 27 Aug 1982  rdr=9
 m dickm how can I be of assistance?
Sent to local user Dick MacKinnon (IBM-CSC)
Ready; 0.06/0.13 2:02pm 27 Aug 1982  rdr=9


 m dickm yes; downstairs...
Sent to local user Dick MacKinnon (IBM-CSC)
Ready; 0.06/0.08 2:02pm 27 Aug 1982  rdr=9
 m dickm I'm going to SEAS too...
Sent to local user Dick MacKinnon (IBM-CSC)
Ready; 0.07/0.13 2:03pm 27 Aug 1982  rdr=9
PUN FILE 0378 FROM IBM-CSC  COPY 001   NOHOLD
 m dickm I'm working on the foils now... I'm not going to talk; just to listen.
Sent to local user Dick MacKinnon (IBM-CSC)
Ready; 0.07/0.13 2:05pm 27 Aug 1982  rdr=10
 rl
```

Figure 3—The Rdrlist fullscreen program is invoked.



```
VM/CMS Console Log

 CCCCCCCCCC      SSSSSSSSSS      CCCCCCCCCC
CCCCCCCCCCCC    SSSSSSSSSSSS    CCCCCCCCCCCC
CC        CC    SS        SS    CC        CC
CC              SS              CC
CC       VV     SSVV     MM     CCMM
CC       VV     SSSSSSSSSSS     CCMM
CC       VV      SSSSSSSSSSS    CCMM
CC       VV       VV    MMSSM  MCCMM
```

```
27 Aug 82 14:16:43
```

```
IBM 3277 Fullscreen Emulation Screen 1
 OHARA    RDRLIST  A0  V 106  TRUNC=106 SIZE=11 LINE=1 COL=1 ALT=22

Cmd   Filename  Filetype Class User    at Node    Hold  Records  Date     Time
      VMSG      GLIEDT   PUN A GLIEDT     WINH6   NONE        1 08/19  06:21:08
      KAY00     NOTE     PUN A KAY00      MSNVM1  NONE       23 08/19  18:27:54
      CKI14     TARVM    PUN A CKI14      TARVM   NONE       18 08/23  17:56:00
*     ZEIGER    NOTE     has been discarded.
 _    R621HLSC  NOTE     PUN A R621HLSC   RALVM6  NONE       18 08/26  11:11:14
      BBB       RCHVM1   PUN A BBB        RCHVM1  NONE       11 08/26  14:01:41
      LE7NTOOL  LAGM1    PUN A LE7NTOOL   LAGM1   NONE        4 08/27  04:44:19
```

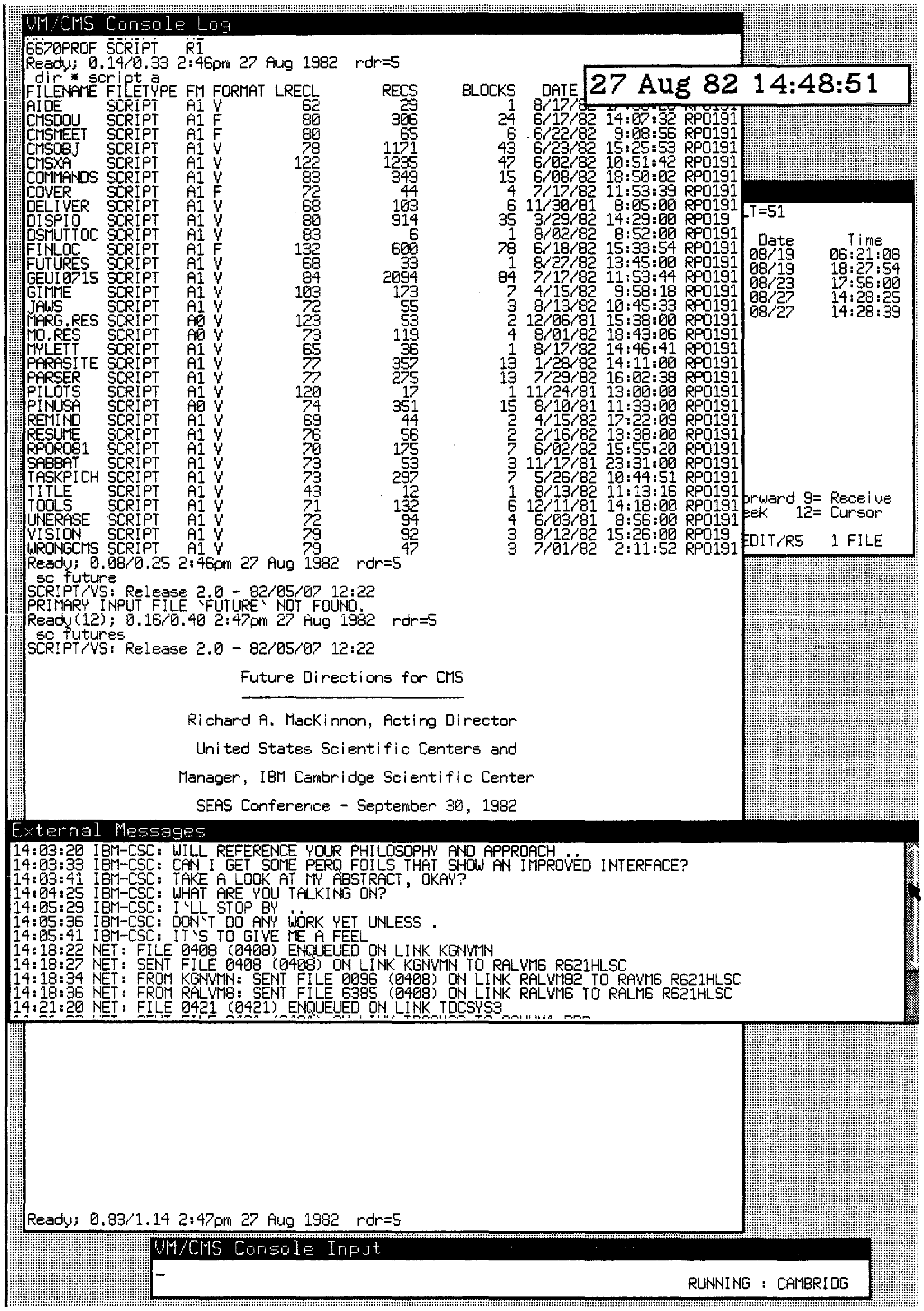
```
IBM 3277 Fullscreen Emulation Screen 2
 7412     PEEK     A0  V 80  TRUNC=80 SIZE=17 LINE=0 COL=1 ALT=17

Note R621HLSC NOTE from R621HLSC at RALVM6.  Format is PUNCH.
* * * TOP OF FILE * * *
Date: 26 August 1982, 10:29:05 EDT
From: H. L. Stuck                  Tie Line 441-6988      R621HLSC at RALVM6
To:   OHARA at CAMBRIDG

A local user has developed a `REPLY` macro.  When you are in NOTE
```

```
3277 PF Keys
PF  1  PF  2  PF  3
PF  4  PF  5  PF  6
PF  7  PF  8  PF  9
PF 10  PF 11  PF 12
```

```
IBM 3277 Fullscreen Emulation Screen 3
 OHARA    NOTE     A0  V 132  TRUNC=132 SIZE=21 LINE=9 COL=1 ALT=0
INPUT MODE:
* * * TOP OF FILE * * *
OPTIONS: NOACK    NOLOG    SHORT     NONOTEBOOK ALL



Date: 27 August 1982, 14:15:17 EDT
From: Robert P. O`Hara          Tieline 249-9274     OHARA    at CAMBRIDG
To:   R621HLSC at RALVM6

REPLY sounds good.  It probably should be coded as an exec (command) so that it
could be invoked from both inside and outside of Xedit.  Why don`t you
advertize it in the next VM newsletter?          Bob

PS. Please send me a copy!_


1= Help     2= Add line    3= Quit          7= Backward 8= Forward 9= =
4= Tab      5= Send        6= ?            10= Split   11= Join   12= Power input
===> * * * INPUT ZONE * * *
                                                          INPUT-MODE 1 FILE
```

```
MARTY    -
ROBMITRO -
SCCPE1   -
SPRINGER -
 REX Exec i
 REX is in
R; T=1.91/3
Ready; 0.05
 m dickm ho
Sent to loc
Ready; 0.13
 m dickm ho
Sent to loc
Ready; 0.06


 m dickm ye
Sent to loc
Ready; 0.06
 m dickm I`
Sent to loc
Ready; 0.07
PUN FILE 03
 m dickm I`
Sent to loc
Ready; 0.07/0.13 2:05pm 27 Aug 1982  rdr=10
 rl
```

Figure 4—The Note command invokes the editor recursively.



```
VM/CMS Console Log
6670PROF SCRIPT    R1
Ready; 0.14/0.33 2:46pm 27 Aug 1982  rdr=5
 dir * script a
FILENAME FILETYPE FM FORMAT LRECL        RECS      BLOCKS   DATE
AIDE     SCRIPT   A1 V         62          29           1  8/17/8
CMSDOU   SCRIPT   A1 F         80         306          24  6/17/82 14:07:32 RP0191
CMSMEET  SCRIPT   A1 F         80          65           6  6/22/82  9:08:56 RP0191
CMSOBJ   SCRIPT   A1 V         78        1171          43  6/23/82 15:25:53 RP0191
CMSXA    SCRIPT   A1 V        122        1235          47  6/02/82 10:51:42 RP0191
COMMANDS SCRIPT   A1 V         83         349          15  6/08/82 18:50:02 RP0191
COVER    SCRIPT   A1 F         72          44           4  7/17/82 11:53:39 RP0191
DELIVER  SCRIPT   A1 V         68         103           6 11/30/81  8:05:00 RP0191
DISPIO   SCRIPT   A1 V         80         914          35  3/29/82 14:29:00 RP019
DSMUTTOC SCRIPT   A1 V         83           6           1  8/02/82  8:52:00 RP0191
FINLOC   SCRIPT   A1 F        132         600          78  6/18/82 15:33:54 RP0191
FUTURES  SCRIPT   A1 V         68          33           1  8/27/82 13:45:00 RP0191
GEUI0715 SCRIPT   A1 V         84        2094          84  7/17/82 11:53:44 RP0191
GIMME    SCRIPT   A1 V        103         173           7  4/15/82  9:58:18 RP0191
JAWS     SCRIPT   A1 V         72          55           3  8/13/82 10:45:33 RP0191
MARG.RES SCRIPT   A0 V        123          53           2 12/06/81 15:38:00 RP0191
MO.RES   SCRIPT   A0 V         73         119           4  8/01/82 18:43:06 RP0191
MYLETT   SCRIPT   A1 V         65          36           1  8/17/82 14:46:41 RP0191
PARASITE SCRIPT   A1 V         77         357          13  1/28/82 14:11:00 RP0191
PARSER   SCRIPT   A1 V         77         275          13  7/29/82 16:02:38 RP0191
PILOTS   SCRIPT   A1 V        120          17           1 11/24/81 13:00:00 RP0191
PINUSA   SCRIPT   A0 V         74         351          15  8/10/81 11:33:00 RP0191
REMIND   SCRIPT   A1 V         69          44           2  4/15/82 17:22:09 RP0191
RESUME   SCRIPT   A1 V         76          56           2  2/16/82 13:38:00 RP0191
RPORO81  SCRIPT   A1 V         70         175           7  6/02/82 15:55:20 RP0191
SABBAT   SCRIPT   A1 V         73          53           3 11/17/81 23:31:00 RP0191
TASKPICH SCRIPT   A1 V         73         297           7  5/26/82 10:44:51 RP0191
TITLE    SCRIPT   A1 V         43          12           1  8/13/82 11:13:16 RP0191
TOOLS    SCRIPT   A1 V         71         132           6 12/11/81 14:18:00 RP0191
UNERASE  SCRIPT   A1 V         72          94           4  6/03/81  8:56:00 RP0191
VISION   SCRIPT   A1 V         79          92           3  8/12/82 15:26:00 RP019
WRONGCMS SCRIPT   A1 V         79          47           3  7/01/82  2:11:52 RP0191
Ready; 0.08/0.25 2:46pm 27 Aug 1982  rdr=5
 sc future
SCRIPT/VS: Release 2.0 - 82/05/07 12:22
PRIMARY INPUT FILE `FUTURE` NOT FOUND.
Ready(12); 0.16/0.40 2:47pm 27 Aug 1982  rdr=5
 sc futures
SCRIPT/VS: Release 2.0 - 82/05/07 12:22

                    Future Directions for CMS
                    _____________________________

              Richard A. MacKinnon, Acting Director

               United States Scientific Centers and

             Manager, IBM Cambridge Scientific Center

               SEAS Conference - September 30, 1982
```

```
27 Aug 82 14:48:51
```

```
T=51

 Date      Time
08/19    06:21:08
08/19    18:27:54
08/23    17:56:00
08/27    14:28:25
08/27    14:28:39

rward 9= Receive
ek    12= Cursor

EDIT/R5   1 FILE
```

```
External Messages
14:03:20 IBM-CSC: WILL REFERENCE YOUR PHILOSOPHY AND APPROACH ..
14:03:33 IBM-CSC: CAN I GET SOME PERQ FOILS THAT SHOW AN IMPROVED INTERFACE?
14:03:41 IBM-CSC: TAKE A LOOK AT MY ABSTRACT, OKAY?
14:04:25 IBM-CSC: WHAT ARE YOU TALKING ON?
14:05:29 IBM-CSC: I`LL STOP BY ..
14:05:36 IBM-CSC: DON`T DO ANY WORK YET UNLESS .
14:05:41 IBM-CSC: IT`S TO GIVE ME A FEEL
14:18:22 NET: FILE 0408 (0408) ENQUEUED ON LINK KGNVMN
14:18:27 NET: SENT FILE 0408 (0408) ON LINK KGNVMN TO RALVM6 R621HLSC
14:18:34 NET: FROM KGNVMN: SENT FILE 0096 (0408) ON LINK RALVM82 TO RAVM6 R621HLSC
14:18:36 NET: FROM RALVM8: SENT FILE 6385 (0408) ON LINK RALVM6 TO RALM6 R621HLSC
14:21:20 NET: FILE 0421 (0421) ENQUEUED ON LINK TDCSYS3
```

```
Ready; 0.83/1.14 2:47pm 27 Aug 1982  rdr=5
```

```
VM/CMS Console Input
_
                                   RUNNING : CAMBRIDG
```

Figure 5—Smooth scrolling of the message window

scrolling allows the data in the window to be read while it is being scrolled, and gives instant feedback as to the direction of scrolling.

CONCLUSIONS

Unfortunately, my sabbatical assignment ended after AIDE had been operationally stable for little over a month. During that time I used AIDE regularly at my desk to access VM/CMS. (I even gave up my regular alphanumeric display terminal!) Based on that short experience the following thoughts are offered.

It is practical to build an improved user interface as a "front end" to an existing interactive system without modifying that system. Such an interface can run existing alphanumeric-display application programs and yet provide a superior user environment.

The large screen is wonderful. In comparison, a standard 24 by 80 display terminal is like trying to view the world through a keyhole.

The font used in AIDE can display more than 90 lines of more than 100 columns of characters on the screen. Yet even this is not large enough. As can be seen in the sample terminal session, the display can become crowded with windows.

Multiple overlapping windows are an effective way to organize the information displayed on a large screen. Displaying additional windows instead of clearing the screen seems to reduce confusion about where one is. Since the screen is never cleared, as so often it would be on a standard display terminal, the need for copying information off of the screen onto paper is greatly reduced.

Pointing devices are a very natural way to select from menus or position a cursor. The use of pointing devices and instantaneous pop-up menus can reduce the need for many commands and modes that are common in most systems.

Areas for Further Study

There are many possible areas for additional work. The following items were planned for AIDE but were not implemented owing to lack of time.

Add additional functions to the windows. It would be nice to shrink windows that are not of current interest to make them occupy less of the display screen. Windows that are inactive might be indicated as such by shading them. Typewriter-mode windows might copy their contents to disk so that a complete session log could be retained.

Provide *soft* program function keys for 3277-based programs. With little change to most programs, the names of the functions performed by pressing the PF keys could be sent to the Perq, and AIDE could thus dynamically label the PF keys. The PF key pad could be displayed and operated as a pop-up menu, thus providing more immediate feedback to the user.

Provide separate windows for system status functions such as mailbox files, terminal and system status settings, current disk links, directory settings, and search paths. These windows would always show the current status of their functions, and a setting could be changed by directly altering the window. This approach is similar to the property and option sheets of the Xerox Star.

Provide graphic displays of system information. Most systems display information such as load, utilization, and search orders as tabular lists. The graphic capabilities of the APA display could be easily exploited to provide this information in a more comprehensible form such as bar charts for load information and tree graphs for directory structures.

ACKNOWLEDGMENTS

This work was done while I was on assignment at the IBM Cambridge Scientific Center in Cambridge, Massachusetts. I would like to gratefully acknowledge Dick MacKinnon for inviting me to the IBM Cambridge Scientific Center, my manager Alex Chandra and my director Dick Kelisky for sponsoring my sabbatical assignment. I also thank my colleagues at the Scientific Center for their advice and support during my stay, especially Sheldon Borkin, John Prager, John Gonzalez, John Webster, Ray Fessel, and Peter Hardy.

REFERENCES

1. Bury, K. F., J. M. Boyle, R. J. Evey, and A. S. Neal. "Windowing vs. Scrolling on a Visual Display Terminal." *Proceedings of Human Factors in Computer Systems,* (March 15–17, 1982), pp. 41–44.
2. Gonzalez, J. C. *Implementing A Window System for an All-Points-Addressable Display,* Bachelor's Thesis, MIT (June 1982).
3. Hartman, J. P. *The TOY System,* Presentation at SEAS Spring Meeting, Noordwijkerhout, The Netherlands, (May 3–7, 1982).
4. *IBM Virtual Machine/System Product: CMS Command and Macro Reference,* IBM Systems Library, order number SC19-6209-1, International Business Machines Corporation, 1980.
5. Lantz, K. A. and R. F. Rashid. "Virtual Terminal Management In a Multiple Process Environment." *Proceedings of the Seventh Symposium on Operating Systems Principles,* (December 10–12, 1979), pp. 86–97.
6. Lipkie, D. E., S. R. Evans, J. K. Newlin, and R. L. Weissman. "Star Graphics: An Object-Oriented Implementation." *Computer Grahpics,* 16, 3 (July 1982), pp. 115–124.
7. McCrossin, J. M., R. P. O'Hara, and L. R. Koster. "A Time-Sharing Display Terminal Session Manager." *IBM Systems Journal,* 17, 3 (1978), pp. 260–275.
8. Meyrowitz, N., and M. Moser. "BRUWIN: An Adaptable Design Strategy for Window Manager/Virtual Terminal Systems." *Proceedings of the Eighth Symposium on Operating Systems Principles,* (December 14–16, 1981), pp. 180–189.
9. Norman, D. A. "Steps toward a Cognitive Engineering: Design Rules Based on Analyses of Human Error." *Proceedings of Human Factors in Computer Systems,* (March 15–17, 1982), pp. 378–382.
10. Ritchie, D. M., and K. A. Thompson. "The UNIX Timesharing System." *Communications of the ACM,* 17, 7 (July 1974), pp. 365–375.
11. Seawright, L. H., and R. A. MacKinnon. "VM/370—A Study of Multiplicity and Usefulness." *IBM Systems Journal,* 18, 1 (1979), pp. 4–17.
12. Tesler, L. "The Smalltalk Environment." *Byte,* 6, 8 (August 1980), pp. 90–147.
13. Yedwab, L., C. Herot, R. Rosenburg, and C. Gross. "The Automated Desk." *Joint Proceedings of the Symposium on Small Systems and the Workshop on Small Data Base Systems,* (October 13–15, 1981), pp. 102–108.

Resiliency of the computerized society*

by REIN TURN
California State University
Northridge, California
and
ERIC J. NOVOTNY
Communications Satellite Corporation
Washington, D.C.

ABSTRACT

Computers support nearly every functional area of a modern society. Consequently, when malfunctions or software errors occur, or when computing support is not available for other reasons, disruptions may result. In networked systems, disruptions may spread and seriously impair an entire functional area of the country. Has computerization produced a new vulnerability for society? Or is society intrinsically resilient and able to absorb large disruptions without grave and lasting societal consequences? This paper analyzes these and related questions. The conclusion is that modern societies (at least the United States) still appear to be resilient. Suggestions for maintaining resiliency are made.

*Based in part, on R. Turn, E. J. Novotny, J. J. Geraghty, E. H. Sibley, and W. H. Ware, *Observations on the Resiliency of the U.S. Information Society* (Arlington, Va.: AFIPS Press, 1982).

INTRODUCTION

In 1951 the first commercial electronic digital computer, the UNIVAC, appeared on the marketplace, and the first step was taken toward a computerized society. Today over a hundred thousand computers in the United States are used by businesses, industry, and government; many more are in private homes as personal computers. Millions of microprocessors are found in pocket calculators, video games, household appliances, automobiles, and other products. In 1981, sales of computer-related products in the United States exceeded $38 billion,[1] and hundreds of thousands of people were employed as programmers, computer operators, or data entry operators. In 1976 it was estimated that over 45% of the U.S. work force could be categorized as employees of the information sector.[2]

It is clear that the United States has already become a computerized society. In such a society, production and distribution of information is central to the economic, political, and social life; and the benefits accrued through the availability of information make the society even more dependent on timely information. This requires extensive automation of information processing, including its dissemination and use in decision-making functions. Examples are management information systems (MIS) and decision support systems (DSS) in business, record-keeping systems in government, automated process control and manufacturing, electronic fund-transfer systems (EFTS), office automation, electronic mail, various reservation systems, and the like. Indeed, it would be difficult to operate a modern society in the manner to which we have become accustomed without computerized information systems.

Strong economic and technological forces continue to exert pressure for increased computerization and thus for increased societal dependency on automated information processing. For example, domestic and international competition is driving computer manufacturers to produce more capable yet less expensive computers, storage devices, terminals, and networks. The need for productivity improvement drives industrial automation. Energy problems accentuate the need for new working patterns, such as working at a terminal at home instead of traveling to work, or substituting teleconferencing for business trips. The increased complexity of management and decision making and of coping with daily life itself call for more information and more capable information processing systems. Computerization of the society continues unabated.

As early as the mid-1960s, when the total computer population in the United States was only 20,000, concern was voiced over the apparent dependence of the society on computer systems.[3,4] The financial industry, manufacturing and process control, and national defense were pointed out as especially critical areas in case of computer failure. Regularly reported in the media were cases of computer crime, violations of individual privacy in computerized record-keeping systems, ineffective and unreliable computer applications, and problems in the design and operation of national command and control systems. Although these were viewed as weaknesses of individual systems, collectively they were beginning to be regarded as a new potential technological vulnerability of the society. Much has been done to correct earlier technological weaknesses, but new ones arise from the very complexity of the new interconnected systems.

Computerization is an international phenomenon and, correspondingly, concerns over societal vulnerabilities that may be created thereby are international in scope. Thus, in 1979 Sweden released the results of a review of the computer vulnerability of the Swedish society,[5] and the Organization for Economic Cooperation and Development (OECD) held an international workshop on the vulnerability of the computerized society in Siguenza, Spain, in 1981.[6] The results of these efforts are discussed later in this paper. It is also clear that the integration of computer and communications systems—*informatics* and *telematics* are the terms used abroad—is regarded as a promising means by which industrial societies as well as developing countries may overcome their economic and societal problems. In France, the report by Nora and Minc[7] observed that "the increasing computerization is a key issue in [the French] crisis and could either worsen or help to solve it." At a meeting on informatics in 1978[8] the developing nations observed that "information is more than power, it is a power system through which other developments will result" and that "any country that is not independent in informatics is not sovereign."

This paper examines the issues that underlie the question of whether computerization increases societal vulnerability, summarizes the results of the Swedish vulnerability review and the OECD workshop, and examines the resiliency and vulnerability of the United States as a computerized society. It is based in part on material prepared by the authors for an AFIPS brief on the resiliency of the information society.[9]

VULNERABILITY FACTORS AND CONCERNS

In common use, *vulnerability* refers to a weakness or a flaw or to a state of being open for abuse, misuse, or damaging actions. *Resiliency* is defined as the ability to rebound after a disruption or setback without lasting ill effects. *Threat* is defined as an occurrence or an action, deliberate or inadvertent, that exploits vulnerabilities to cause undesirable effects.

Threats can be gradual or indirect as well as sudden or direct. In the context of the present analysis the postulated vulnerability is the computerization of society: the possibility that *because* of using computers, there is an increase of threats that may cause significant short-term disruptions, losses, or crises; or long-term adverse changes that affect a very substantial part of the society in question. Thus, the size of the society is an important consideration, as is the geographical area this group inhabits. For example, the power blackout in the northeastern United States a few years ago affected a region and a population greater than quite a few countries in Europe, but it was not viewed as a nationwide disruption in the United States.

Generic Factors

Before the threats that may exploit the computerization of society can be examined, a taxonomy of societally adverse effects is needed. However, an objective taxonomy is difficult to produce, since an effect that may be judged adverse by one group in the society may be judged beneficial by another. The following list suffers from this problem and therefore must be regarded only as illustrative; the effects listed are not in any particular order of severity or importance, nor is the list complete:

1. A severe disruption of national economy and large losses that are only partially recoverable.
2. A paralyzing disruption of the functioning of the national financial system.
3. Severe disturbance of the daily life, and of the ability to obtain the necessities of life, of a large part of the population, with attendant social unrest.
4. A large decline in the productive capacity of an important industrial sector and/or a large fraction of the work force.
5. A drastic decline in the standard of living for the entire population or large subpopulations.
6. Massive wasting of resources, with associated severe shortages in sectors where these resources should have been used.
7. The institution of governmental policies that have harmful effects on large parts of the society.
8. The erosion of citizens' constitutional or human rights.
9. Increased dependence on foreign powers in economy, finance, or politics.
10. Substantial decline of the educational level of large segments of the population.
11. Decline of individual values of the society toward a lack of interest in personal or societal advancement.
12. War or nuclear conflict.

Is it possible for computerization to cause directly, or to contribute substantially to, the occurrence of some of these harmful effects? If not, computerization should not be regarded as a potential nationwide societal vulnerability but, at most, as one that might have local or regional effects on society or individual organizations. Indeed, there are computer applications that support societal functions that, in turn, either could cause harmful effects or could be subject to such effects themselves. Examples of such computer applications are the following:

1. Command and control systems for national defense and for weapons control.
2. Real-time process control systems (manufacturing, transportation, power distribution, telecommunications).
3. Distribution of merchandise, food, raw materials, and supplies.
4. Record-keeping systems of personal information about individuals.
5. Financial and fund transfer systems, such as EFTS.
6. MIS and DSS for strategic planning; computer-based models.

The threat mechanisms that may cause these computer applications to contribute to societally harmful effects may include the following:

1. Sudden loss of computers in large numbers, crippling an entire societal function for a prolonged period of time. Such a loss may occur as a result of large-scale failure of electrical power, destruction from hostile or terrorist actions, subversion, contamination of data or programs in large networked systems, or labor strife.
2. Capture by terrorists of critical computer systems in an operational state.
3. Subversion of computer systems to sabotage their functioning systematically so as to cause widespread adverse effects or so as to cause incorrect decisions to be made because subverted systems are being used.
4. Accumulation in databases of erroneous information that cannot be identified or located and that over time renders such systems unusable (if this condition is discovered) or its products incorrect.
5. Accumulation of program flaws that eventually produce false results, causing them to become untrusted and unused if this condition is discovered.
6. Overdependence, on a large scale, on the use of computer systems (for example, to reduce below critical level the number of people who could perform the automated tasks manually or who know how they should be performed).
7. Deliberate misuse of the computer system for societally detrimental purposes by owners or users; misuse for fraudulent purposes and personal gain.

The above lists attempt to lay a foundation for assessing computer applications in a given society to determine whether the society is vulnerable or resilient. The assessment methodology is still to be developed, but certain risk assessment techniques have been proposed.[10]

The Swedish Vulnerability Review

In the case of a society considerably smaller than the United States—Sweden—a preliminary review of societal vulnerability was made by a Ministry of Defense committee, SARK, from 1977 through 1979.[5] The premise of the SARK review was that a "technologically developed society cannot do without ADP," and its conclusion was that (for Sweden) the "vulnerability is unacceptably high in today's computerized society." This conclusion was reached by an analysis of the following "vulnerability factors," which are similar to the "threat mechanisms" discussed earlier:

1. Criminal acts involving computer systems—espionage, sabotage, crimes against property and resources, and terrorism with a political aim that in the future may become a means of aggression between states apart from conventional wars.
2. Misuse for political purposes—threats of economic sanctions, which may in the future be an increasingly common means of pressure to attain political aims. (E.g., when a country is highly dependent on imports of computer equipment or services, a blockade against their import, as well as of spart parts, may very quickly have serious effects.)
3. Acts of war—including the loss of parts of interconnected systems to the enemy; destruction of systems; requirements for curtailing data processing for civilian purposes; and problems with obtaining equipment, spare parts, or services from abroad. Electromagnetic pulses from nuclear detonations far away may destroy or damage computers and communications systems.
4. Catastrophes or accidents—due to natural causes, explosions of hazardous materials, fire, or external events. Power outages can produce lengthy disruptions of ADP service.
5. Registers containing information of a confidential nature—vulnerable to misuse of information about, or to bringing undesirable pressure against, the data subjects.
6. Functionally sensitive systems—used for financial activities, production management, process control, and the like, where misuse or attacks may result in drainage of resources, reduced production of goods, disruption in transportation, or other problems that may have important economic effects.
7. Concentration—geographic or functional, where accumulation of computer systems in a particular geographic area may make them an attractive target and increase damage from attacks or catastrophes; centralization of function which may make an entire functional area subject to severe disruption as a result of a single attack or catastrophe. Dispersion and decentralization can reduce these vulnerabilities.
8. Integration and interdependence—the existing information flows between systems, including sharing of databases or processing functions, which creates situations where disturbances in or loss of one system propagates to other systems. Accidentally or deliberately introduced erroneous data in these systems can quickly spread throughout the system.
9. Accumulation of large quantities of data—large compilations of data items that may be insensitive themselves but may offer new intelligence-gathering opportunities when accumulated.
10. Deficient education—inadequate education with respect to vulnerability and security, resulting in a lack of awareness of potential problems.
11. Low quality of hardware and software—faults in equipment or errors in software that may lead to accidents immediately or that may cause long-term damage that may be hard to detect and impossible to repair.
12. Key persons in computer operation—system engineers or programmers, who often develop systems that only they can understand, thus putting the users into a dependent position. Dissatisfied employees have damaged systems or planted program changes that can disrupt a system at a later time. Key persons themselves are vulnerable to actions against them as a means of disrupting systems.
13. Documentation—lack of adequate documentation of system changes, causing problems to surface when changing the system, personnel, or operational requirements; unnecessary costs are accrued in terms of system unavailability or loss of productivity.
14. Emergency planning—failure to plan adequately for packup and resumption of data processing in case of disasters or in emergency situations.
15. Dependence on foreign sources—the need for components, spare parts, and services from abroad, which makes Sweden dependent on an undisturbed flow of trade. Existing supplies would last for only a brief period of time were they interrupted by war, blockade, trade embargo, or the like.
16. International data flows—causing security and vulnerability problems of dimensions other than those existing under purely national conditions. If data processing [for a country] is done in another country or on another continent, and if input and output are to pass through several countries, the misappropriation risks of various kinds increase. Protecting against events abroad is naturally more difficult than building up a domestic system of protection.

The SARK review concludes that the vulnerability of existing systems (in Sweden) can be limited and that by assessment of vulnerabilities it should be possible to build new systems in a different, more resilient way. This could be done by providing adequate information and consultation for some cases, but in other systems "more far-reaching measures, including licensing" would be needed. In 1981 Sweden established a new group, the Vulnerability Board (SARB) to study the question further.

The OECD Workshop Findings

The OECD workshop on computer vulnerability examined three themes: (1) protection of computer and telecommunication installations, (2) design and management of information installations, and (3) societal dependence on information technology. The first theme has been treated extensively in literature[11, 12] and is not central to this paper. The other two themes are of interest, however. In the design and management of information systems the following issues were identified:

1. The need for research on the question whether there are limits to the areas or functions of society where computers should be used.
2. The need to make contingency planning an integral part of the system design process.
3. In information systems with international scope, the need for a legal framework to cover nontechnical aspects of the vulnerability issue.
4. Effects on vulnerability of the increased complexity of systems and of the software maintenance problem.
5. The apparent lack of computer auditing in many organizations, which represents a major risk and thus implies a potential vulnerability.

At the OECD workshop, *vulnerability* was defined as "the possibility of loss, injury, or denial of equal rights to a significant segment of the population, the weakening of social stability, or the risk to national sovereignty due to dependence on computer-based information technology." The following issues were identified that involved societal dependence on information technology:

1. Citizens' inequality of access to computerized media, databases, and knowledge necessary for using information machines.
2. The need for anonymous methods for effecting commercial transactions in computerized systems to protect citizens from (1) overzealous or malicious statistical data collections and (2) construction of profiles of personal traits by tracking or inference.
3. Ways must be found for protecting consumers against misuses of computer programs that compute shopping bills or debit bank accounts. A parallel to "inspectors of weights and measures" must be developed for use in these systems.
4. The need to discover measures to prevent the growth of new monopolies based on computer systems.
5. The apparent lack of knowledge by nontechnical people of what computers and computer networks are capable of doing, which may be an underlying source of society's computer vulnerability. All citizens should be provided the opportunity of adequate training in the use of computers as a part of their basic education.
6. The importance of educating decision makers about the potentials, limitations, and actual use of computers to reduce the danger of misinterpretation of computer-produced results.
7. The need to investigate the cultural effects of computerization more thorougly (e.g., possible effects on family life, thinking patterns, or social behavior).

The workshop concluded (1) that computer and communication technologies represent positive developments that contribute to social and economic progress, but (2) that the question facing policy makers is how to ensure that the extensive use of these technologies occurs to the best advantage of society while minimizing disadvantages or dangers to it. The consensus was that the OECD is a suitable international forum for studies and exchanges of experience in this area.

SOCIETAL RESILIENCY

Resiliency is the ability to absorb disruptions and damages without suffering long-lasting or irreversible ill effects. In the present context it is the ability to recover from failures of computerized systems or from their misuse without lasting, societywide ill effects such as described in the previous section. The following factors appear to enhance resiliency in a computerized society or a modern computerized nation:

1. Geographic and demographic aspects of the society—territorial size, population size and distribution, and size and geographic distribution of industries and commerce activities supported by computerized systems. Large societies tend to be more resilient.
2. The degree of multiplicity in providing societal functions and services and the amount of redundancy in supporting systems—Societal functions served by multiple, overlapping, competing, decentralized, and heterogeneous organizations and systems are more resilient.
3. High levels of functional-level and society-level preparedness; contingency planning; and establishment of tested backup and recovery agreements, procedures, and systems will enhance resiliency.
4. Maintenance of domestic capabilities or nondisruptable channels to reliable foreign suppliers of data processing equipment, spare parts, maintenance support, or services is important for resiliency in computational support systems.
5. Preservation of "corporate memory" of how functions were performed and supported prior to computerization; and deliberate maintenance of adequate manual backup capability in organizations, as well as in entire functional areas, is necessary for resiliency.
6. A population used to a technological environment with hands-on experience with technical devices, and literate in computer and communications technology, tends to be inventive and innovative in disruptions and crises. Thus this population is a very important contributor to societal resiliency in technical matters.
7. Continuous monitoring of new developments and systems and of the degree of computerization from the vulnerability/resiliency point of view is important for resiliency.
8. Legal safeguards against misuses of computerized systems—such as those for privacy protection and against computer abuse—and legal requirements for examining the vulnerability/resiliency aspects of new com-

puter system applications provide essential foundations for achieving and maintaining societal resiliency.
9. A national information policy that includes the achievement of an acceptable level of resiliency of the computerized society is required for resiliency.

The vulnerability factors of the previous sections and the resiliency factors listed above should be used together to assess the degree of vulnerability and the degree of resiliency of a society. Applied to Sweden by the SARK committee, the conclusion was concern over Sweden's high degree of vulnerability. The same factors applied to the United States, for example, appear to produce a different result.[9, 13] There are differences in the size of the society, in the multiplicity of nearly all business, industrial, and societal activities; in dependence on foreign sources for data processing equipment or support; in possibilities of foreign invasion, and the like. The geographical and geopolitical situations of the two countries are vastly different. The United States' society at the present time appears to be reasonably resilient regarding lasting detrimental effects of failures of computer systems, even on a large scale, or disruptions of computer-based services as in financial systems, even for an extended period of time. There may be transitory local or regional problems and losses, but not nationwide effects or losses.

More specifically, using the vulnerability factors identified by the Swedish SARK committee, the following observations can be made about the resiliency of the computerized U.S. society:[9]

1. Computer crime or computer abuse cases have occurred in the United States with substantial losses to individual businesses, but their true extent is not known. It is not likely that computer crime will grow to cause significant societywide losses, even if targeted against EFTS. Resiliency will be provided through advances in computer-communications security techniques.
2. Political pressure on the United States by curtailing imports into the United States of computer equipment or services is not an issue at present. But one must note that many U.S. computer manufacturers operate foreign-based manufacturing and assembly plants. A well-orchestrated action by several foreign countries would be required to cause a significant disruption.
3. Computers are an integral part of the U.S. military systems and are critical for national defense. Nuclear explosions generate electromagnetic pulses which, when coupled into computer communications systems, can damage circuitry.[14] Countermeasures against EMP effects include shielding and the use of fiberoptics.
4. Natural catastrophes in the U.S. will be localized and cannot cause system outages that will cause nationwide disruptions.
5. Privacy protection in personal information "registers" is provided by the Privacy Act of 1974 on the Federal level, and in some 13 states. Much less protection is provided in the private sector. Enactment of the recommendations of the Privacy Protection Study Commission[15] would substantially increase resiliency in this area. However, continued monitoring of new proposals for collecting or interchanging personal information is required lest the present rather minimal resiliency be lost.
6. Many computer-based systems in the United States may be regarded as functionally sensitive: transportation, process control, telecommunications; distribution of power or goods and services, banking, law enforcement. But each area includes numerous independent providers of services, and simultaneous loss of all is highly unlikely. As in the case of labor strife, some region may be inconvenienced for a while, but no nationally impairing disruption is likely. With reduction of service providers and integration of systems this prognosis may change, however. Functional-area contingency and backup plans are needed for improved, continued resiliency.
7. Concentration of computer systems into a few locations is a possible vulnerability. In the United States there are many computers in large cities. Power outages can disrupt their services and cause regional problems. The trend toward distributed systems is alleviating this factor, as is the increased acquisition of emergency backup power supplies by private and governmental organizations.
8. Integration and interdependence of systems tend to be increasing, but they are not nationwide in scope (except certain systems such as the air traffic control, interbank funds transfer systems, and various reservation systems). Technical means can be used to guard against accidental or deliberately induced deadlocking of these systems, and against spreading of erroneous data. Caution must be exercised in the design and operation of these systems.
9. Accumulation of very large databases of personal or business information raises privacy concerns and possibilities of (indirect) manipulation. There is a latent danger under some scenarios, but resiliency is provided by the United States' open society and democratic system of government.
10. Lack of security awareness continues to be a problem in the United States, especially in private sector systems; and only a few organizations are taking steps to improve security features or security awareness. This may permit more incidents of computer crime.
11. Defects in hardware occur and cause malfunctioning of individual systems or networks of systems, but technical means are available and are increasingly applied to reduce this problem. However, it is imperative that certain critical systems be protected against erroneous behavior caused by hardware malfunctions or software errors.
12. The capability of key personnel in computer facilities, such as system programmers, operators, and certain applications programmers, to disrupt computer operations maliciously, or misuse systems, continues to be a problem internationally. Personnel procedures and

technical means are available, and should be used, to ensure that no single employee is in absolute operational control of a critical computer system.

13. The importance of adequate documentation of computer system and software designs and implementations is well known. Computer models and simulations used in designing other systems or in supporting decision making must be well documented and show explicitly any simplifying assumptions or aggregations that may have been made. Improper use of models may indeed produce serious societal impacts, especially if long-term policies are involved.
14. The importance of emergency planning is self-evident. It must be expanded from organizational-level planning to functional-level planning to increase societal resiliency.
15. The United States tends to be a supplier rather than a user of data processing equipment or services abroad. However, U.S. manufacturers own and operate manufacturing facilities abroad and use parts manufactured by foreign vendors. Thus, there is an element of dependence and there are possibilities for the disruption of supplies. Though this vulnerability factor should not be ignored and should be monitored, it is not seriously undermining resiliency at this time.
16. The question of international data flows is of great importance to the United States. Many multinational corporations are U.S. business firms with operating units in many countries. Restrictions on data flows can interfere with their operations and adversely affect U.S. commerce. Such restrictions may result, in part, from rigorous application of foreign data protection laws.[16] Such restrictions have not yet been applied to any significant degree, but the U.S. support of the doctrine of free flow of information is not shared by countries that perceive an imbalance of data flows and economic losses to their domestic data processing industries.

The above observations strive to support in qualitative—perhaps even in intuitive—terms the claim that the computerized U.S. society is still resilient. But a systematic, quantitative analysis is needed to answer the question credibly.[17,18]

CONCLUDING REMARKS

A computerized society is not necessarily a vulnerable one. Indeed, the United States appears to be resilient to all but large-scale hostile actions. But there is nothing in this conclusion that ensures any permanence for this apparent resiliency. Deliberate efforts are required by all sectors of the society—the general public, the government, and the information processing profession and industry.

The Need for Public Awareness

Complex technologies tend to engender myths that sometimes obscure the reality. Nuclear technology is a case in point. Incidents such as the Three Mile Island nuclear power plant accident jolt people into opposing their technology, even though these incidents do not cause damage. They create visions of future highly damaging catastrophes. The increasing use of computer technology, likewise, may lead the public to believe that malfunctioning or loss of an important computer center could cascade throughout a computer network and affect the entire country. Local disruptions in computer systems are certain to occur and thus reinforce the expectation of a major computer-based or computer-generated disaster. Even if such a major event does not occur, public confidence in all computer or telecommunication systems may erode. Should the event actually occur (e.g., an airliner crash due to malfunctioning of the air traffic control system), the public reaction may lead to drastic control of the data processing profession and industry.

Improved computer literacy among users and the general public can foster a more realistic, less mystified attitude toward computer technology and its applications and develop a healthy skepticism regarding the power of computer technology and a realistic understanding of its limitations. For this purpose it is important to study and report on all computer-related disruptions, to explain the roles of computers and their uses in these incidents, and to distinguish human errors from machine-induced errors. It is equally important to explain and publicize the capabilities and roles of computer systems in preventing disruptions from occurring in computer systems themselves or in other systems. These tasks are largely the responsibilities of the information processing profession, especially the professional societies.

Information processing professionals must also accept the responsibility for another element of public awareness—continuous monitoring of the computerization process for any technological or societal disruptions or losses, studying these for symptoms of the erosion of resiliency, and ensuring that computer systems and applications are designed with societal resiliency in mind.

Governmental Obligations and Opportunities

A major governmental obligation is to include the resiliency question as an important part of the overall information policy of the United States. Thereby it can be made an important system design criterion. This approach has already been proved effective in the area of protecting individual privacy; no one would today propose a personal information record-keeping system without considering privacy protection and data security.

Another important role for the Federal government is setting and promoting standards on information resiliency for the information processing field as a whole, as well as individual functional areas of the society. In addition to focusing attention on the need for resiliency and providing guidelines for resiliency improvement, publication of standards also serves an imporant educational goal. The Federal government can also contribute by supporting research and development efforts on vulnerability topics. Some results arise naturally (1) from defense-related programs involving system survivability, fault tolerance, backup and recovery techniques and (2) from the work on trusted/secure systems and software.[19,20] Other results will require specially targeted research efforts.

Finally, should large-scale disruptions of information-based economic functions actually occur, the government may have to assume a role in providing assistance and relief, as it does already in the case of natural disasters. The objective of such relief would be to restore the interrupted service quickly (e.g., by making available information processing support) and then to assist in reestablishing the service. Further studies of this concept are needed, however.

Private Sector Opportunities and Obligations

Most of the responsibility for information resiliency in the United States lies in the private sector. User organizations, as well as providers of information systems or services, must decide on reasonable protective measures and be willing to pay for them. Backup and recovery plans and exercises, data quality control checks, and data security controls are the best insurance against adverse events. Taken collectively, these actions enhance societal resiliency.

Further improvement of resiliency requires industrywide cooperation. Organizations should work to ensure continuity of critical societal functions by establishing systemwide contingency planning, backup, and recovery. Industry associations have the important role of promoting such cooperation, as well as promoting industry-sponsored research and development in resiliency techniques.

In summary, the United States apparently has in place (perhaps as a vestige of the manual operation age) an infrastructure that provides societal functions and controls the necessary information services. At present this structure can provide an acceptable degree of resiliency against failure of information systems that support such functions. However, this structure is not so robust as to continue providing resiliency indefinitely. It is imperative that the information systems now in operation be evaluated from time to time concerning their impact on and contributions to resiliency and that maintenance of resiliency be an important design criterion for new systems.

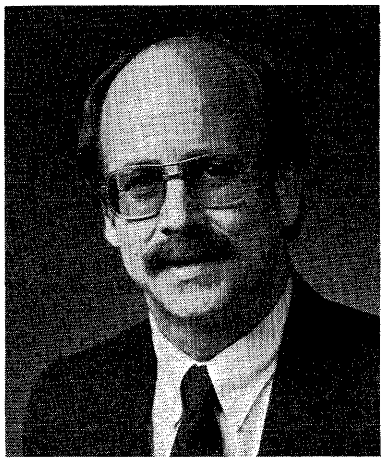
ACKNOWLEDGMENTS

The authors are grateful for permission from the AFIPS Governmental Activities Committee (J. Gosden, chairman) to use in this paper material from the AFIPS Brief *Observations on the Resiliency of the U.S. Information Society,* prepared by the AFIPS Panel on the Resiliency of the U.S. Information Society (R. Turn, chairman). The authors, who are also the principal authors of the brief, acknowledge important contributions to the brief by panel members or advisers E. H. Sibley of Alpha Omega Group; W. H. Ware of The Rand Corporation; D. Brandin and D. B. Parker of SRI International; J. Geraghty of the IBM Corporation; S. H. Nycum, Esq., of Gaston, Snow, and Eli Barlett; A. Roth, Esq., formerly of the AFIPS Washington Office; and R. G. Canning of Canning Publications.

REFERENCES

1. "World Markets Forecast for 1982: U.S. Markets." *Electronics,* January 13, 1982.
2. Dolatta, T. A., M. I. Bernstein, R. S. Dickson, N. A. France, B. A. Rosenblatt, D. M. Smith, and T. B. Steel, Jr. *Data Processing in 1980–1985.* New York: John Wiley & Sons, 1976.
3. Weiss, H. "The Weeks the Computers Stopped." *Datamation,* April 1967.
4. "Is the Computer Running Wild?" *U.S. News & World Report,* February 24, 1964.
5. Ministry of Defense, Sweden. *The Vulnerability of the Computerized Society: Considerations and Proposals.* (Translation.) Stockholm: The Ministry, December 1979.
6. "Workshop Stresses Dependence on Comptuers." *Transnational Data Report,* July/August 1981.
7. Nora, S., and A. Minc. *The Computerization of Society: A Report to the President of France* (Translation of 1978 Report). Cambridge, Mass.: MIT Press, 1980.
8. "SPIN Conference Resolutions." *IBI Newsletter,* No. 27. Rome: International Bureau of Informatics, 1978.
9. Turn, R., E. J. Novotny, J. J. Geraghty, E. H. Sibley, and W. H. Ware. *Observations on the Resiliency of the U.S. Information Society.* Arlington, Va.: AFIPS Press, 1982.
10. Hoffman, L. J. "Impacts of Information System Vulnerabilities on Society." *AFIPS, Proceedings of the National Computer Conference* (Vol. 51), 1982, pp. 461–467.
11. Hsiao, D. A., D. S. Kerr, and S. E. Madnick. *Computer Security.* New York: Academic Press, 1979.
12. Turn, R. (ed.). *Advances in Computer System Security.* Dedham, Mass.: Artech House, 1981.
13. Kirchner, J. "AFIPS Finds U.S. Not Vulnerable in DP Disasters." *Computerworld,* June 28, 1982, p. 23.
14. Lerner, E. J. "Electromagnetic Pulses: Potential Crippler." *IEEE Spectrum,* May 1981, pp. 41–46.
15. Privacy Protection Study Commission. *Personal Privacy in an Information Society.* Report of the Commission (D. Linowes, Chairman). Washington, D.C.: Government Printing Office, 1977.
16. Turn, R. (ed.). *Transborder Data Flows: Privacy Protection and Free Flow of Information.* Arlington, Va.: AFIPS Press, 1979.
17. Zientra, M. "Risk Analyses Said Needed to Determine Technology's Effects." *Computerworld,* June 28, 1982, p. 27.
18. Brandin, D. H. "The Horse or the Herring." *ACM Communications,* September 1982, pp. 597–598.
19. Walker, S. T. "The Advent of Trusted Operating Systems." *AFIPS, Proceedings of the National Computer Conference* (Vol. 49), 1980, pp. 655–665.
20. Turn, R. "Private Sector Needs for Trusted/Secure Computer Systems." *AFIPS, Proceedings of the National Computer Conference* (Vol. 51), 1982, pp. 449–460.

OFFICE AUTOMATION


Daniel J. Drageset
Atlantic Richfield Co.
Los Angeles, California

The Office Automation track will focus on critical issues related to the implementation and effective use of this rapidly emerging technology. The track will begin with an overview of the technology and the current state of the art, which will be of value to the uninitiated as well as to the practitioner. This will be followed by sessions dealing with pressing issues for the practitioner or manager: organization, implementation, and productivity. Additional sessions will focus on specific aspects of the technology, concentrating on where we are today and where we are going, emphasizing requirements based on experience. The track will conclude with a vision of where the technology is headed and its potential to significantly change the way knowledge workers work and extend their effectiveness.

Interfacing people with their machines

by NANCY B. FINN
Boston University
Boston, Massachusetts

ABSTRACT

As computer technology proliferates, the techniques applied to the interface of people with their machines will assume great importance if the large dollar investments in equipment are to be recouped in the form of increased productivity.

This paper propounds the theory that change in any form is difficult for people to accept, especially in the office environment, and that it is essential to involve the work force in the change to automated information processing if new technology is to succeed.

The paper outlines several ways to accomplish this, including polls and individual interviews with all employees before the system is purchased and the design of a public relations effort to help employees understand the applications of automation to their personal roles. The paper also covers various ergonomic questions related to computer systems, such as noise, heat, illumination, static, terminal design, and general environmental questions.

The issue of accommodating people in an office to office automation is a complex one, requiring a concerted effort on the part of all managers involved with the implementation.

INTRODUCTION

The implementation of office automation in a company is a change far different from any that has preceded it. Unlike the opening of a new plant, the shifting of a division, the reorganization of a department, or the implementation of a specific new procedure that attempts to accomplish a single goal, automating the office changes an entire information processing scheme, altering the daily tasks and chores of individuals and rearranging not only people's lives but the physical design of an organization as well. In other words, office automation, if properly implemented, ultimately restructures the office environment, requiring a shift in management tactics as well as in management procedures.

With automation, new roles and responsibilities are created, forcing office workers to reshape the patterns of their day and to assume new, specific chores related to the computer systems they must now learn to use. The familiar tools of the typewriter and the telephone are shunted aside, and workers are told to learn to operate these terminals through short training sessions and by struggling with complex documentation. They often find themselves moved from private cubicles to an open space, where they share a terminal with other office workers. Personal relationships between secretarial workers and managers are dissipated as managers are forced to reassign the workload and in some cases are required to keyboard original documents themselves. In theory, the task is to use the finest applications of computer technology to increase productivity. In reality, when dealing with human beings, it does not often work easily.

IS CHANGE A POSITIVE OR A NEGATIVE?

Research in human behavior has clearly indicated that human beings react poorly to change, though in varying degrees.

Studies have shown that humans view change with a mixture of fear, ambivalence, lack of interest, and downright unwillingness to modify their behavior. People resist change because they fear they will not be able to develop the new skills and behavior required. They resist change because it is different from the established patterns with which they have become comfortable. Some groups have even been known to engage in direct action to block the implementation of something new.

If automation is to succeed—if it is to fulfill its promise of increasing productivity and making information processing a manageable, viable facet of office operations—then management must develop a basic understanding of what it takes to keep the elements of change under control.

THE OFFICE STRUCTURE

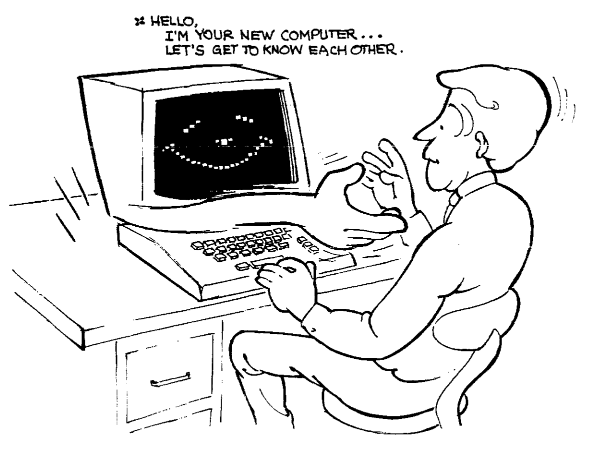
Most offices are structured with an executive/manager at the top and several managers reporting, each in hierarchical steps, to the person above. Secretaries and clerks are assigned to managers at various intervals in the spectrum.

Sociological studies have shown that there are two levels of jobs in an office environment: primary jobs, or those that offer substantial pay, opportunities for training and advancement, considerable supervision, and a degree of job security; and secondary jobs, which offer low pay, little training or advancement, little job security, and constant supervision from someone above. Implementation of office automation will have an effect on both primary- and secondary-level jobs, since information flow in the hierarchy is accomplished step by step, with very little direct information contact between the bottom levels and the top. Messages are passed from one level to the next in an extremely cumbersome fashion, and both primary- and secondary-level employees are involved in the process.

A COMPUTER ON EVERY DESK

In the early 1950s there were approximately 1,000 computers in America and no microcomputers, microprocessors, or chips. By 1976, there were more than 220,000 computers and three-quarters of a million microcomputers in use. By 1980 Americans were using over 10 million microprocessors. It is estimated that the market for intelligent terminals will grow at a rate of 25% each year throughout this decade.

With the placement of terminals on the desks of virtually every manager and secretary in the office environment, information flow will change radically. Vocal and telephone communications, face-to-face meetings, and many written mes-



sages currently processed will disappear. In their place will be shared databases, common electronic mail/messaging systems, and common files that will open up new avenues of communication among the individuals in an organization. Each individual user will be able to supervise more closely the dissemination of his/her messages. New pathways of communication will be established. Interpersonal roles will change as well. Clerical workers will become an unknown entity as such highly routine work as filing, mail sorting, mail delivery, and voluminous copying tasks are handled electronically by computer systems that perform these tasks by the mere push of a button or the command of a voice.

Out of the labor force will come a group of paraprofessionals who will assume semi-management positions in the office, using computer technology to monitor and control a variety of processing chores currently performed manually. Managers and executives will use computer conferencing to replace time-consuming travel. They will have decision support systems to handle budget preparation, sophisticated modeling tasks, and maintenance of their calendar and to give them access to a variety of databases.

Knowing that this is ahead, workers at all levels of the office hierarchy are fearful when they hear that office automation is coming. How to implement automation with a minimum of dissent and disarray is the question that must be addressed if electronic processing in the office is to meet with success.

INFORMATION MANAGEMENT IS EVERYONE'S CONCERN

The first hurdle to be passed when office automation is considered is for an office to take a close look at its information processing and determine where, when, and how automation might fit into the scheme. This information management study must be undertaken with all members of an office staff participating, not just a selected few, as has been the practice in the past. To insure that the information management study is handled properly, a team of individuals should review office procedures as they relate to the following:

1. The kinds of documents typed or handwritten
2. Filing procedures and how the files are organized
3. Sorting and retrieval of documents
4. Copying chores and the numbers of copies generated
5. Mail volume, reception, and distribution
6. The volume of telephone calling and how calls are channeled
7. Final document preparation: Is it via Xerox, mimeograph, phototypesetting?
8. Budget preparation, access to budget information, manipulation of data
9. Planning and forecasting tasks
10. The allocation of time to these tasks by executives, managers, secretaries, clerks, typists, bookkeepers, etc.

The results of the study must present an overall picture of the nature of the work being done daily in the office. It must determine how technology can be used most effectively to improve productivity and bring about beneficial change.

In the information management study, each member of the office staff should be individually interviewed and asked to analyze items to be included in the review as they relate to his/her job. During the same interview, individuals should be asked for opinions about the impending change and should be encouraged to express any of the fears and anxieties about automation they may have as well as to express any helpful ideas.

For the sake of efficiency, especially in a large office, a questionnaire might be used to complement personal interviews. The questionnaire will provide more extensive feedback on each individual's role in the information processing scheme than the interviews alone will reveal. They should not, however, replace the interviews. (See Figure 1.)

Many companies have totally dispensed with an information management study, charging the computer department with the task of evaluating needs and determining equipment configurations. Generally, computer departments require that individual offices justify their requests for a system by maintaining a daily work log for each individual. The informal discussions encouraging employees to voice their feelings about automation are ignored. In their place, hostility builds among both managers and clerical workers who are being asked to account for their time without receiving a proper explanation of why or how the information is needed. This intrusion into work operations is considered an invasion of privacy by office workers. As a result, office automation is resented before it ever has a chance to prove its worth.

There is reason to believe that change is coming. Centralized data processing departments are slowly losing clout in large organizations. No longer are companies looking to their corporate data processing managers to provide the leadership needed to bring computing power out of the back room and into the hands of all employees. Steering committees whose members come from all levels of the corporate hierarchy are assuming this role, insuring that each faction in the company is represented.

In offices where automation has previously been installed, but where a proper information management study was never completed and where employees are disgruntled and grumbling about their computers, a study should be undertaken. It is never unreasonable to review information processing practices and attempt to formulate an organized, sensible, and easier-to-use database. Although it is more difficult to convince already unhappy employees that automation has a positive aspect, the problem is not unsolvable. Involving employees in change at any step in the process is better than not including them at all.

AUTOMATION DOES NOT EQUAL JOB LOSS

Another way to avoid negative reactions to office automation is to dispel the myth that automation in and of itself will create job loss and unemployment. Automation generally has not resulted in mass layoffs in most companies. Although it is true that the elimination of personnel is one factor to cost-justify office automation, reducing the work force usually takes place through attrition rather than through wholesale firing. Initially, computers have even made it necessary to increase a work force, since massive information reorganization must occur when a database is input. Once a computer system is up and working, employees generally find that they are freed from the burdens of paper work and many routine tasks, enabling them to perform administrative work and to participate in decision making more effectively. Thus computers have not eliminated jobs but have made them more interesting, challenging, and fruitful.

A PUBLIC RELATIONS PLAN

Companies that have most successfully brought automation to the office environment while maintaining harmony and good will among the office workers in the initial stages have embarked on an active public relations program to market this concept to their employees. All companies that are automating can benefit from a plan of action incorporating all the media of communication (including words, visuals, and other tools) to keep open channels of communication and to create acceptance of office automation as a beneficial tool. This mammoth effort to launch and maintain an aggressive public

Figure 1

Sample Questionnaire

1. Could you please analyze your work day into segments and indicate the percent of time you spend with the following activities:

Handwritten work	________
Typing	________
Mail sorting	________
Mail answering	________
Telephone usage	________
Copying	________
Filing activities	________
Retrieval	________
Calendar Maintenance	________
Meetings	________
Calculating	________
Reading/review	________
Original work	________

2. Are you in favor of office automation?

Yes	________
No	________

3. Would you personally use a computer:

Yes	________
No	________

4. Please rank in order those chores where you feel a computer would help you the most:

Typing	_____
Filing	_____
Retrieval	_____
Elimination of telephone calls	_____
Elimination of meetings	_____
Original work	_____
Budget maintenance	_____
Calendar maintenance	_____
Record maintenance	_____

5. Do you produce finished (typeset) documents?

Yes	_____
No	_____

6. Do you feel that a computer will help you become more efficient at your job?

Yes	_____
No	_____

7. Can a computer terminal be placed at your current work-station?

Yes	_____
No	_____

8. Have you ever used a computer terminal?

Yes	_____
No	_____

relations campaign will pay off in positive employee attitudes and in the increase in productivity that is being sought through computing power.

The public relations plan should include several facets. Most companies have working communication media, such as house organs and weekly newsletters, that can feature explanatory general articles about the applications of office automation tools. In addition to established publications, companies might want to issue an office automation bulletin or newsletter, biweekly or monthly, with articles of specific interest—for example, news about new software packages and contributions from employees about new applications for their computer terminals.

Another tool that could be used is bulletin boards, which offer a good place to corroborate information with brief messages, appropriate posters, and drawings. A dynamic bulletin board, effectively designed, with imaginative posters, cartoon drawings, and well-displayed notices on office automation, can be a considerable asset in propounding the idea that automation is a concern of all members of the organization.

Information racks (installed in many companies primarily for morale and employee education) can provide easy access to written materials describing office automation tools. These racks should be placed in lunchrooms and corridors so that everyone has access to them.

A common device in large corporations, which has received favorable reception, is the use of telephone newslines to link managers with various employees in the hierarchy. Establishment of a newsline update on office automation is another way of keeping employees informed.

Displays of equipment, set up before a computer system is installed, may make employees feel more familiar with terminals. Even the use of closed-circuit television in large organizations where it is often difficult to reach everyone is a helpful feature in communicating the need for and the importance of office automation.

Most essential in the public relations effort is the establishment of regular departmental meetings involving all the employees in open discussion about the changes and the uses of computer systems. This is the most important element in the attempt to assuage ruffled feelings and emotions and remove a sense of alienation among employees, who often feel that they are not consulted about major changes in the office where they spend the largest part of their day.

ERGONOMICS

Ergonomics is the science that studies the relationship between people and machines. It focuses on the design of computer terminals and the ease with which the user is able to adapt to computers, including a concern for structural and physical design elements. All the public relations strategies and attention to information management before a computer system starts operating will be fruitless if installed systems do not solve the user interface problems that have hampered the industry in the past. Some of the problems that have to be addressed are discussed in the following sections.

Noise

The increasing acquisition of computer equipment, along with new office design encouraging an open landscape, has resulted in an environment that is not conducive to concentrated work. The noise generated by clacking printers and the ringing of telecommunications devices is an irritant to workers whose enclosures have been removed in many offices, as well as to workers who are in close to a printer. The use of acoustic panels to absorb distracting sounds and the installation of plush carpeting have provided partial solutions to the problem. Improvements in printer technology are slow in coming; but there will be a gradual change, which should alleviate the problem.

Static

Computers generate static electricity that can make an office an uncomfortable place to work. Antistatic mats and sprays are a partial solution to this irritant. Vendors are working to eliminate some of the static at its source.

Illumination

The placement of a computer terminal near adequate lighting seems to be another problem. The installation of lights in panels and other pieces of furniture designed for the automated work station promises to help in that situation.

Heat

Computer terminals also emit heat and must be placed in an atmosphere that is neither too hot nor too cold. Adequate air conditioning must be planned for and provided.

Design

The general design of the office, and especially the chair and table used by the terminal operator, is extremely important for comfort. A recent trend in vendors of smaller systems has been to provide terminals that are set on tables and contain the disk drives so that all users buy their furnishings with the system.

As computer technology proliferates in the office, the ergonomics of a system will be yet another determinant in its acceptance by the workforce. The ability of an individual user to adapt easily to the equipment and to accept it as a productive tool depends on ease of use. Attention to the elements of noise, static, illumination, heat, and design is essential to an appropriate interface.

CONCLUSIONS

Acceptance; use; and, most important, the establishment of a new pattern for information processing in the office will be realized only if new technology is implemented with concern for the human reaction to change. Even if all the creativity and imaginative selling technique that a company puts into the promotion of a new product is directed to the selling of computer power to their employees, the system will not succeed unless employees are directly involved in bringing about the changes that will come with office automation.

The most sophisticated management studies in human behavior have revealed that the greatest productivity booster is the willingness of employees to give 8 hours of work for 8 hours of pay. By accepting as a given that employees will not welcome change with open arms, but need to be introduced to it tactfully and gradually; by accepting as a given that management must include the employees in the decision-making process; by accepting as a given that a thorough understanding of information management is an essential first step toward the implementation of computer power; by accepting as a given that an aggressive campaign to influence, to educate, and to inform all employees about office automation is important to its adoption—organizations can fulfill the promise of electronics.

BIBLIOGRAPHY

1. Cribbin, James J. *Leadership, Strategies for Organizational Effectiveness.* AMACOM, 1981. New York: American Management Association, 1981.
2. Cutlip, Scott, M. and Allen H. Center. *Effective Public Relations* (5th ed.). Englewood Cliffs, N.J.: Prentice-Hall, 1982.
3. Finn, Nancy B. *The Electronic Office.* Englewood Cliffs, N.J.: Prentice-Hall, 1982.
4. Yankee Group. *Office Automation: The Human Dimension* (Volume X, CIS Planning Service). Yankee Group, Boston, Mass., July 1980.
5. Zachman, William F. "Key To Enhancing System Development Productivity." AMACOM, 1981. New York: American Management Association, 1981.
6. Kotter, John P., and Leonard A. Schlesinger. "Choosing Strategies for Change." *Harvard Business Review,* March–April 1979, p. 106.
7. Nolan, Richard L. "Managing Information Systems by Committee." *Harvard Business Review,* July–August 1982, p. 72.
8. Judson, Arnold S. "The Awkward Truth About Productivity." *Harvard Business Review,* September-October 1982, p. 93.
9. Zuboff, Shoshana. "New Worlds of Computer-Mediated Work." *Harvard Business Review,* September-October 1982, p. 142.

Current issues in electronic mail—Heralding a new era

by WALTER ULRICH
Walter E. Ulrich Consulting
Houston, Texas

ABSTRACT

Electronic mail is a quick, convenient, and low-cost way to handle people-to-people communications. Each form of electronic mail—computer message systems, communicating word processors, facsimile, and voice mail—is successfully serving organizations and individuals. The ultimate integration of these forms of electronic mail will be far more useful functionally than each one taken independently. This paper discusses the integration of electronic mail and introduces three companion papers; the four together make up the 1983 session on electronic mail.

INTRODUCTION

Electronic mail is approaching its 162nd anniversary. Each stage in the development of electronic mail has enhanced communications and made the world a smaller place. This was true in 1844 when the telegraph was first demonstrated, and it is true today.

Over the years, electronic mail has moved closer to the individual:

- The experiments of the 1820's served a handful of the scientific intelligentsia.
- The telegraph office of the 1850's served a community.
- The teletype of the 1920's served an organization.
- The computer message of the 1970's served a department.*
- Today, computer message systems and voice mail systems are serving individuals.

Once only remotely available, electronic mail is now accessible to individuals on their desks and in their homes. There are several forms of electronic mail, including computer mail, communicating word processors (CWP), facsimile, and voice mail. Advances in any one over the last 12 months could fill a volume.

Throughout its first era, electronic mail has moved closer to the individuals. Technically at least, personal electronic mail is a reality. Socially, it is becoming better accepted and more prevalent. The era of personalization has arrived.

In 1983, however, we stand at the threshold of an important development in electronic mail. Various forms of electronic mail are becoming integrated. We will be able to prepare, transmit, and receive messages that combine the elements of text, image, and voice as required. Separately, each form of electronic mail is suitable for only selected applications. Together, integrated electronic mail can handle a broad array of business communications.

This paper serves as the introduction for the 1983 National Computer Conference Session on Electronic Mail. Current issues in electronic mail are discussed and the new era of integrated electronic mail is described.

WHAT ROLE FOR ELECTRONIC MAIL?

Electronic mail is the transmission of message text by electronic means. By definition, electronic mail is well-suited for one-way transfer of information. Other forms of communications are better suited for interactive communications, as Table I indicates.

Traditional methods of one-way communications have been either slow (the U.S. Postal Service) or inconvenient (like internal memos and TWX/Telex).[1] Therefore, two-way communication methods have been misused for one-way information transfers. Half of all telephone calls could be replaced by an appropriate message system.

The total cost of a message transaction includes its handling at both ends. Computer-based electronic mail makes it possible to process messages as well as exchange them. For some companies, electronic transmission costs are already competitive with the first-class stamp.[2,3] When you include the economies from easier message handling at both ends, the economies in favor of electronic mail are even better. Add to those benefits the immediacy of electronic delivery, and the justification for electronic mail is overwhelming. According to a recent projection by the Office of Technology Assessment,[4] which is excerpted in Table II, electronic-related mail will reach 23 billion pieces in 1990 and 86 billion pieces in the year 2000.

Electronic mail is a quick and inexpensive way to transmit information. It complements advances in data processing, data management, and office systems. Computer processing is getting 25% to 30% cheaper every year, and electronic mail rides free on this fortunate technological wave.[5] Its potential is to displace half the business telephone calls and a third of the traditional mail volume over the next two decades.

Types of Electronic Mail

Different types of electronic mail serve different purposes. Computer mail (also known as computer-based message systems) is designed to provide a text message facility for executives and professionals. Communicating word processors

TABLE I—Types and uses of forms of communication

Communication Technology	One-Way or Two-Way	Example
First class mail	One-way	Requesting information
Teletype	One-way	Placing an order
Telephone	Two-way	Clarifying complex information
Meeting	Two-way	Negotiations
Electronic mail	One-way	Requesting or providing information

*Throughout the 1970's high technology companies and the ARPA project team used computer mail for personal messaging. However, many commercial users would establish an electronic mail box for departments and branch offices rather than for individuals.

TABLE II—Projections of growth in types of mail*

Type of Mail	Examples	Volume in Billions 1990	Volume in Billions 2000
Conventional mail	Letter	109	75
Computer-assisted	E-COM	9	13
Electronic funds transfer	Telephone bill payment	5	25
End-to-end electronic mail	Computer mail, teletext	9	48
Total volume		132	161

*Chart is adapted from the OTA report on Electronic Mail and Message Systems.[4] See page 38 of the report for details of the assumptions.

(CWPs) allow secretaries to transmit formal correspondence and documents. Both computer mail and CWPs send the message content as digitally encoded characters. One character takes one byte.

Facsimile sends the page image. Each page is broken down into a number of spots. The number of spots varies from 6,000 to 160,000 spots (or picture elements—pixels) per square inch, depending on image quality. The spots then are sent as an analog signal or as a stream of bits indicating whether the spot is black or white.†

With voice mail, the speech waveform is encoded into its digital representation. 32,000 to 64,000 bits are typically needed to encode one second of speech. Stored on disk, the message can be sent to one or more recipients, edited, or saved for later use. The natural inflection and emphasis is preserved.

Each form of electronic mail is best suited for specific purposes. Table III reviews these.

THE NEED FOR INTEGRATION

Modern business is complex, as the following scenario indicates. Owen, the president, asks Jack for a review of one competitor's new product for next month's planning meeting. Jack, the marketing vice-president, requests that Melinda, who is in charge of competitive analysis, estimate the sales volume and compare its features with their company's top-of-the-line product. He asks Howard, the chief engineer, to estimate the production cost. Finally, he asks Joe, the western region sales manager, if the product is being pitched to a key customer in Los Angeles. Except for Joe, each of them assigns all or some part of the task to subordinates.

Jack gets a formal report from Melinda, a text message from Howard, and a voice message from Joe. With his administrative assistant, he prepares a formal report for Owen and transmits copies to the other attendees of the upcoming planning meeting.

†Modern facsimile machines will use numerical methods to reduce (compress) the actual number of bits transmitted.

TABLE III—Appropriate uses of electronic mail (EM)

Type of EM	Format	Application
Computer mail (CBMS)	Text	Messages between professionals and administrators. Mostly informal. Requests for complex information. Providing information with figures.
Communicating word processors (CWP)	Text	Formal memoranda and correspondence. Lengthy reports and documents.
Facsimile	Text	Pictures and charts. Other information already prepared but not in digital format.
Voice mail	Speech	Short messages. Requests for simple information. Confirming travel plans and meeting schedules. Simple instructions and brief answers to requests for information.

Throughout the process, numerous verbal and written comments and messages are exchanged. Any one style of communication by itself would be insufficient. Multiple styles are required.

THE INTEGRATION OF ELECTRONIC MAIL

The new era of electronic mail will provide that integration. Graphs and charts will be routine parts of text messages.[6] Documents will be annotated with voice comments, and voice messages will refer to text.[7] Data in computer messages will be brought into formal documents being prepared on word processors.

Office requirements will go beyond integrating electronic mail. Spread sheets from personal computers and information in public and private databases will also become a transparent part of the information infrastructure.

Even control mechanisms will become integrated. The integrated message facility (IMF) will provide for a single interface for controlling all message facilities and the telephone system. For example, a window on the user's workstation will provide a complete status for that user on

- The telephone (incoming calls, camp on, queueing, etc.)
- Text and voice messages (electronic in-basket)
- Operator messages
- Secretarial intercom.

The workstation to support that user will provide transparent interface to all the communications facilities needed. The basic integrated terminal telephone (ITT) is shown in Figure 1. For many users, the workstation will meet the individual's personal computation and data-storage needs as well. And in some cases, powerful graphics will be available.

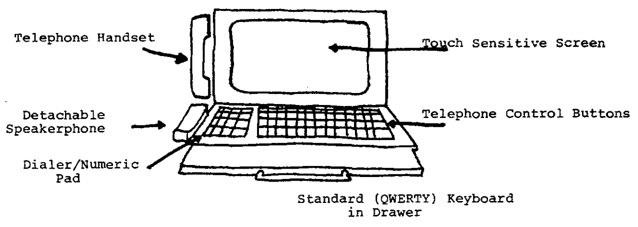


Figure 1—Basic integrated terminal telephone

Various workstations have their proponents. Will the office of the future be populated by Tymshare Scanset terminals, Osborne "portable" computers, Wang's Alliance, Xerox Stars, or whatever? There is no one answer. Everybody's needs are different and computerization makes it possible to tailor the workstation to the user.

CURRENT ISSUES IN ELECTRONIC MAIL

This paper introduces the session on electronic mail. This session focuses on the most critical issues in electronic mail for 1983.

Voice mail first became available early in this decade. Voice Mail International is one of only a few vendors of voice mail systems. They have focused on providing store-and-forward voice systems to meet specific application needs. By targeting applications, they expect to demonstrate measurable benefits. A companion paper in this volume describes voice-mail systems and discusses the state of the art of this newest form of electronic mail.

Most computer-based electronic mail systems are used within a company, and in the continental United States electronic mail spans four time zones and three thousand miles. When looked at in the perspective of world-wide commerce, four time zones are insignificant. Europeans are home from work before their counterparts in California eat breakfast. The Yankee Group has been a colorful and controversial observer of technological change. In a companion paper in this volume, one of their consultants discusses both intercompany and international considerations in electronic mail.

Electronic mail has been viewed as character-oriented throughout most of its history. Only recently have voice implications been seriously considered. The perspective on electronic mail of a company with traditional grounding in voice communications is valuable indeed. This is specially true when that company has recently renounced its traditional roots in favor of the brave new world of the future. American Bell, born by Caesarian section from AT&T, has promised to bring together enhanced message and data services. The third companion paper is by their executive director of research and development. His focus is on the integration of electronic mail.

Collectively, this introduction and the three related papers represent the state of the technology in 1983. The justification for electronic mail is no longer in doubt. Personal electronic messaging is indeed available in text and in voice. Integration of function is the promise yet to be fulfilled in this decade.

ELECTRONIC MAIL IN THE NEW ERA

Electronic mail will provide the user with powerful voice, text, and image functions. Compound messages will include elements of each. The workstation will be a multipurpose device and act as the window to all communications and information services. The proper electronic mail combination(s) will always be available to meet the specific requirement(s) at hand. These facilities will be used in ways we cannot predict, but they will cut costs and increase productivity. This will happen so fast and yet evolve so naturally that we will hardly realize what happened.

REFERENCES

1. Holden, J. B. "Experiences of an Electronic Mail Vendor."*AFIPS Proceedings of the National Computer Conference* (Vol. 49), 1980, pp. 493–497.
2. White, J. W. "Texas Instruments Computer Communication Network and its Support for the Automated Office." *AFIPS Proceedings of the National Computer Conference* (Vol. 49), 1980, pp. 515–526.
3. Taylor, H. D. "HP Communication System." *National Telecommunication Conference Record 77* (Vol. 3), 1977, pp. 21:6-1–21:6-5.
4. Office of Technology Assessment. "Implications of Electronic Mail and Message Systems for the U.S. Postal Service." GPO stock number 052-003-00885-8, September 1982.
5. Noyce, R. N. "Microelectronics." *Scientific American*. 327, September 1977, pp. 63–69.
6. Hasiuke, K., K. Konishi, T. Asami, and A. Kurematsu. "Text and Facsimile Integrated Terminal." *National Telecommunications Conference Record 80* (Vol. 3), 1977, pp. 60:5.1–60:5-5.
7. Maxemchuk, N. F., and H. A. Wilder. "Experiments in Merging Text and Stored Speech." *National Telecommunications Conference Record* (Vol. 1), 1980, pp. 16.1.1–16.1.6.

The integration of multimedia communications

by B. P. DONOHUE, III
American Bell, Inc.
Lincroft, New Jersey

ABSTRACT

Effective integration of multimedia communications requires a unified electronic work station. The unified work capabilities that must be provided are information access translation, information processing, information storage, and information movement. This paper describes the current issues involved in providing these unified capabilities. Examples of current multimedia communications systems are also given.

INTRODUCTION

The challenge of the next generation of communications systems is to maximize user access to the myriad of information sources and information processing in a cost-effective and user-friendly manner. Meeting this challenge will require a unified multimedia electronic work capability for knowledge and clerical workers. The purpose of this paper is to review the major issues in providing such a unified work capability that spans voice, text, and graphics and that is capable of improving productivity in the office.

Multimedia communications are considered here in the context of four generic user work-station functions as well as specific applications such as voice messaging and electronic mail. Descriptions of these generic functions and specific applications are given in the next section. This is followed by a discussion of the issues in the integration of multimedia communications for each of the four generic work-station functions. Finally, some current examples of multimedia communications systems will be described.

ELECTRONIC WORK STATION FUNCTIONS

Work Station Overview

The electronic work-station functions to be considered are shown in Figure 1. Note that the work station uses four generic capabilities: information access translation, information processing, information storage, and information movement. Note also that both control and user information can cross the user interface in multimedia form. Specific electronic work functions are built from these generic capabilities. These electronic work functions will now be described in the context of current examples and the desired multimedia communications capabilities that arise naturally from them.

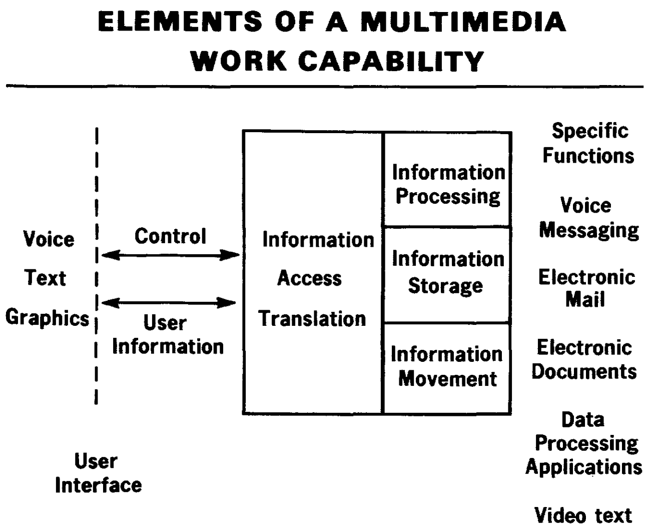


Figure 1—Electronic work station functions

Voice Messaging

The telephone call is the dominant form of electronic communication today, and it is likely to remain so for many years to come. The telephone call has been viewed by the user as a single-medium information movement process. However, this process is rapidly changing with the introduction of information processing and storage capabilities. Typical of the capabilities being added are the ability to send voice store and forward messages or to leave voice messages for busy or unanswered telephones. An early service example of voice store and forward capability was the Bell System Customer Calling Trial.[1] A currently available voice store and forward service is the Voice Mailbox Service (VMS) of ECS Telecommunications, Inc. Today voice mail capabilities are also available in customer premises systems that come with user control options allowing message management based on a rich set of commands. The commands typically allow message editing, filing, retrieval, sending, and deletion.

The second major way in which telephone communication is changing for the user is by the addition of data and control capabilities to aid in making the telephone call. Examples of these capabilities include station sets with programmable function keys for automatic number selection and data terminals with data entry capabilities for directory functions.

Voice messaging can be usefully integrated with all of the remaining work functions to be described. The remaining work functions are performed primarily in text or graphical form, so multimedia integration is required.

Electronic Mail

Electronic mail or electronic messaging systems exist in a wide variety of public and private network forms today.[2,3] They are designed to accept text input, provide text editing functions, and allow message distribution and finally message reception and filing. Today electronic mail systems are used as separate text-only systems. In the future it will be desirable to integrate them in at least three ways.

First, the control information from electronic mail and voice messaging could be coupled so that a message received in one form could be easily responded to in another. For example, a useful command to accompany the reading of electronic mail would place a voice call to the originator of the text message. A related capability might allow a secretary to use electronic mail to create memos of telephone calls and then provide automatic dialing of the party that left the message.

The second way of integrating electronic mail and voice messaging would be to couple user information so that composite messages can be created. For example, an electronic text message could be annotated with a voice message and forwarded to a third party.

The third form of integration of electronic mail and voice messaging could be through media conversion or information access translation. An example of this is speech synthesis of a text message. Translation from speech to text is also desirable, but will be a more limited capability for some time to come. Hence this translation will probably be limited to small vocabularies such as control information. An example of such an integrated translation would be the use of verbal commands to scan messages on an associated text display.

Electronic Documents

Electronic documents are larger-volume text and graphics units that are handled differently than electronic mail today because of their size. The document is often created in electronic form through word processing and then may or may not be proofread and distributed electronically. There is a clear trend towards all-electronic distribution and filing.[4,5] As this trend progresses there will be an increasing need for multimedia integration. For example, voice annotation of text will be an effective way of proofreading documents. Note that at this point electronic documents and electronic mail will likely share many of the same multimedia integration needs.

Data-Processing Applications

The electronic work station requires access to a wide variety of data-processing applications. These applications are usually tailored to the business using them and private electronic records of such things as financial data, customer data, or product data. These data are usually managed through a database facility.

Today data-processing applications are usually dealt with in a text-only form. However, there are a number of ways multimedia communications can increase the effectiveness of the application. One example is to allow simple voice retrieval of information stored in text form. Such information access translation using Touch-Tone or voice recognition for input commands can greatly increase the number of terminal devices (telephones) capable of accessing a data processing application.

Videotext

Videotext is an inexpensive form of graphical information access that was initially targeted at residential customers. Recently, however, there has been increasing interest in it for internal business use.[6] The primary use of videotext is likely to be database access and simple information entry. As such it will be another form of communication desirable in the electronic work station. The introduction of videotext capabilities will require additional information access translation capabilities.

ISSUES IN MULTIMEDIA INTEGRATION

Referring back to Figure 1, it should be noted that the specific functions to be integrated are all available today in some form. What is missing is a unifying framework for user interfaces, information access translations, information processing, information storage, and information movement. The central problem in multimedia communications is to provide such a unifying, yet robust framework. Each aspect of the framework will now be examined separately.

User Interface

The user interface is the aspect of multimedia communications that will determine whether the added automation actually increases productivity. An integrated user interface must allow easier access to multiple functions than separate interfaces do if it is to increase productivity.

A prime requirement for the integrated user interface is that there be a common command structure across functions. For example, users should not have to learn *n* different ways to delete an item if they are accessing *n* different functions. Work on this issue is in progress.[7] In addition to a common command structure, it is desirable to present the user with menus. Menus allow the user to easily select the next action from a predefined set and thereby reduce the number of commands whose details (such as command structure) need be remembered. Examples of the use of menus are given in.[7,8,9,10,11]

A second major requirement for the user interface is that it allow the user to work on or see information from multiple sources simultaneously. On a display screen this is often called windowing.[7,12] It is important because many work functions require the user to access two information sources and then process them together. For example, to fill out one electronic form, it may be necessary to read another electronic file. Ideally, the relevant information from the file would be transferred to the form without retyping. The ability to do such multitasking is needed even for such simple jobs as reading a document. For example, two parts of a document may have to be compared in order to obtain complete comprehension of it.

The multitasking referred to above is also required for multimedia communications. For example, if a display is used both to receive electronic mail and to provide telephone directory information for subsequent calls, the user may want to see both types of information simultaneously.

A third requirement for the user interface is that it provide integrated status information across multiple functions. For example, a message-waiting indication should be provided across all messaging systems.

Information Access Translation

The information access translation capability of an integrated multimedia system will have to be rich if it is to allow access to the wide variety of information systems available. This is well known in the data or text-communications area, where protocol conversion is a standard problem. By comparison, in the voice-communications area there used to be relatively few standards for transmission and signaling. Digital systems such as voice messaging are changing that, since many use special digital voice encoding to store voice messages. These digital encoding schemes are often not directly compatible and require conversion back to an analog signal to send messages between systems. This is undesirable, as the goal in the future will be to make each of the four information-handling building blocks digital.

A second major issue in the area of information access translation is the need to effectively support single-medium terminals as well as multimedia terminals. Even when multimedia terminals can be readily purchased, there will be a large installed base of single-medium terminals that need to access a given application. That application must, therefore, work well with both types of access.

Information Processing, Storage, and Movement

The functions of information processing, storage, and movement are considered together, because they all involve the same central problem of multimedia integration. The central problem is that while all of the media forms discussed here can be represented digitally, that is, by bit strings, the processing, storage, and movement technology most cost effective for each of them may be different. This is most clearly seen by way of examples.

Voice and data (text) information movement can both be accomplished by digital transmission facilities. However, the error performance required by the two information types is different. Voice information can accept modest error rates because of the natural redundancy of speech. Data or text information, however, must have an extremely low error rate, since each letter or number must be correct (especially in financial transactions). The result of this is that the most cost-effective systems for voice and data are not the same.

Voice and data information can also both be stored digitally in a database. However, the desired database operations to be performed on these types of information and the basic sizes of the records are usually not identical. Again this leads to the result that the most cost effective systems for voice and data are not the same.

There are other illustrations of this issue that can be given. The important point, however, is that cost considerations will mean that multimedia systems may have coupled processing, storage, and movement, rather than a single underlying technology that supports all media in the same way.

CURRENT EXAMPLES OF MULTIMEDIA COMMUNICATIONS

Successful integration of voice and data (text) terminals has been demonstrated. A series of efforts at Bell Laboratories[8–15] illustrates what can be done using a data terminal tightly coupled with a voice terminal. The data terminal serves to provide voice-terminal control functions while at the same time providing access to data-only applications. This system has been successfully demonstrated in a number of settings, including the executive office.[14]

The ability to have coupled simultaneous voice and data channels is becoming available in a number of PBX products. This provides part of the basis for effective integrated multimedia communications. As described in this paper, the challenge is to provide a unifying foundation of user interfaces, information access translations, information processing, information storage, and information movement.

REFERENCES

1. Bergland, G. D., E. R. Kerkeslager, R. J. Nacon, and G. W. Smith, Jr. "New Custom Calling Services." *International Switching Symposium '79 Conference Record,* Paris, France, May 1979, pp. 1256–1262.
2. Ulrich, W. E. "Introduction to electronic mail." *AFIPS, Proceedings of the National Computer Conference* (Vol. 49), 1980, pp. 485–488.
3. Panko, R. R. "A Survey of Electronic Message Systems." *Proceedings of the Pacific Telecommunications Conference,* 1981, pp. A3-1–A3-10.
4. Schick, T., and R. F. Brockish. "The Document Interchange Architecture: A member of the family of architectures in the SNA environment." *IBM System Journal,* 1982, pp. 220–244.
5. IBM Distributed Office Support System General Information, GH12-5124, IBM Corporation Document.
6. Davis, D. B. "U.S. businesses targeted as major videotext market." *Mini-Micro Systems,* Sept. 1982, pp. 145–151.
7. Zloof, M. M. "Office-by-Example: A business language that unifies data and word processing and electronic mail." *IBM System Journal,* 1982, pp. 272–304.
8. Thompson, R. A. "Accessing Experimental Telecommunications Services." *Proceedings of the National Telecommunications Conference,* 1981, pp. F2.2.1–F2.2.5.
9. Thompson, R. A. "User's Perceptions with Experimental Services and Terminals." *Proceedings of the National Telecommunications Conference,* 1981, pp. F2.6.1–F.2.6.5.
10. Allen, R. B. "Cognitive Factors in the Use of Menus and Trees: An Experiment." *Proceedings of the National Telecommunications Conference,* 1981, pp. F2.5.1–F2.5.5.
11. Smith, D. L., and R. D. Gordon. "An Access Tree Editor." *Proceedings of the National Telecommunications Conference,* 1981, pp. F2.7.1–F2.7.5.
12. Schell, W. M. "Control Software for an Experimental Teleterminal." *Proceedings of the National Telecommunications Conference,* 1981, pp. F.2.3.1–F.2.3.5
13. Hagelbarger, D. W., R. V. Anderson, and P. S. Kubik. "Experiments with Teleterminals." *Proceedings of the National Telecommunications Conference,* 1981, pp. F2.1.1–F2.1.5.
14. Klavan, R. N. "Enhanced Communications in an Executive Office." *Proceedings of the National Telecommunications Conference,* 1981, pp. F2.4.1–F2.4.5.
15. Bergland, G. D. "An Experimental Telecommunications Test Bed." *Proceedings of the National Telecommunications Conference,* 1981, pp. F2.8.1–F2.8.5.

Voice mail

by PAUL F. FINNIGAN
Voicemail International, Inc.
Santa Clara, California

ABSTRACT

If you think 1982 brought us revolutionary advances in communications technology, then hold on to your hats—1984 is just around the corner, and the growth of the voice store and forward market will start to prove everyone was right: The voice mail race is on. There have been survivors and losers—the real winners will be the users. Vendors that don't understand the dynamics of customer acceptance and don't engineer a usable method of applying voice mail will lose the fundamental opportunities of this technology. Voice mail is not as complicated as it looks. You pick up any telephone, anywhere in the world, and record a message for someone else in your own voice. It's a new medium of communicating. Voice mail is a fast, convenient way to tell someone something when you can't reach them directly. Everyone will use voice mail.

In the very near future every telephone will have voice recording. You'll be able to call a friend, and if the line is busy, or if no one answers the telephone, you'll be able to touch a key, hear a tone, record a message, and hang up. When your friend gets home, he'll go to his phone and touch a key to have the message played back.

For the last four years I have been able to go to any telephone and send a voice message to anyone in the world.

The potential for voice store and forward is just beginning to be realized. The Yankee Group once reported it would be a 500-million-dollar-a-year industry by 1985. Last year, less than $10 million was realized in voice store and forward services and systems. Who is going to make the projections we've read come into focus? Where are the opportunities?

The opportunities are real. The projections are understated. The movement has begun. Name any business—they are using or will be using voice mail. If you have a telephone, you need voice mail. Voice store and forward is a product of the computer industry; another computer technology, another type of computer looking for ways to be used.

After sitting through several national symposiums and seminars on electronic mail, you realize that about half the speakers dwell on voice store and forward. The PABX, data processing, paging, electronic mail, and voice mail companies display genuine ingenuity linking voice mail services to every conceivable office-of-the-future use that one could imagine. Projections of increased productivity and cost savings—and their respective quantitative methods, for the most part—ignore end users' perception of voice mail and the communications dynamics resulting from it.

It is surprising when you look at the vendor list. There's a healthy representation of heavyweights who've diversified into voice mail. It's even more surprising that less than 100 systems were installed last year, and only a few thousand companies were using voice mail services*—not a significant number compared to those who should be active users. Their needs are to improve communications and reduce telecommunications-related costs. Economic realities are going to drive those needs to greater levels. That's the opportunity.

Large systems, small systems, dedicated systems, general systems, and networked, interfaced, linked, expandable systems will be rolling off the assembly line, presumably on a direct line to eager customers. The demand for services these computers provide will be there. Functionality will be more streamlined. But will people understand what voice mail is? Can anyone say they have a grip on its dynamics, beyond pushing the buttons on the phone?

*According to presentations at a Probe Research seminar New York, September 15–16, 1982.

We have a rule at Voicemail International: "Don't tell anyone about voice mail; show them." Understanding when to use voice mail is paramount. Plumbers, real estate people, lawyers, bankers, doctors, pilots, stockbrokers, and lonely hearts have understood. Through an international voice mail network, thousands are using voice mail worldwide. We have found it extremely important to show them when, and where, they will use these services.

On the other hand, we have found thousands of people who won't use voice mail. We've performed a considerable amount of research examining this phenomenon. The microphone syndrome and the "Rockford syndrome" give us two glimpses into the way people perceive what voice mail is and why they hesitate at first even to consider its use. Behavioral considerations are fascinating: Many of the measurable criteria can be directly correlated to what has been found in the usage characteristics of other electronic mail users. Many differences have been witnessed in the acceptance patterns of various voice mail users.

The microphone syndrome occurs when you call a number and someone tries to put a microphone in front of your mouth. One company that risked its existence, and lost, tried to introduce voice recording into a medical answering service. Most callers needed to leave messages that were difficult to talk about, even in a real-time conversation: For example, Mrs. Jones, who wasn't feeling very well at all, called the doctor and was unexpectedly asked, by a computer, to leave a recorded message. This was something new, different, and difficult compared to what she had been doing. She didn't like it.—Isn't voice mail supposed to be convenient and user-friendly? In this application, it wasn't.

Give the same person the means to leave a message for her husband—whose office phone is always busy or rings without an answer—and you have an application that works. She expects the recording service; and she is not speaking to a professional office or a stranger. By dialing the star button twice, she can leave a message in her own voice for later pickup by her husband. She doesn't own a system, she's not a subscriber to any voice mail service, and she's not calling a telephone-answering type of number—she's just making a call over a telecommunications carriers' lines whose switch has a specialized voice mail enhancement. The carrier provides the capability, the convenience, and the means to use voice mail—when she needs it, not when someone else wants to make her use it.

The "Rockford syndrome" is named after the TV series, "The Rockford Files." A phone-answering machine is in the background. James Garner, in the company of a woman, listens to a caller recording a message. A significant number of people have the impression that when you record a message, uninvited ears will be able to listen. In the business world, the political consequences of passing messages on to someone else can be devastating to one's future.

There are many situations where voice mail is just not applicable. But often voice recording and delivery capability can bring what you want to the party. Voice mail that provides specific solutions for specific problems has shown most clearly its benefits in terms of cost reduction, time saving, and time recapture.

Understanding end users' needs and identifying who can best benefit from using a voice mail service is the point to start at. Then consider the large number of people who have general and specific needs to obtain information.

One example would be airlines; some supply flight information, vacation tips, automated reservation services, and many travel-related services toll-free to callers through the specialized services a voice mail network can provide. Within the airlines themselves, flight crew scheduling, flight attendant scheduling, cargo information, and emergency notifications can be provided by the same voice mail computer.

One major airline uses voice mail to tell its reservations centers that its main computer has slowed down—in effect, a specialized computer tells people that another specialized computer isn't working. Numerous applications in the computer and data processing industries are handled by voice mail services.

Voice mail is ideal in the service industries. A service person can receive scheduling assignments by voice mail while on the road, even directly from the customer who needs service immediately. The service dispatcher can keep track of who is doing what and can get up-to-the-minute reports on the status of the calls their service people are making. Voice mail improves communications effectiveness whenever you have people on the move and their activity creates value.

Voice mail can give engineering changes and delivery reports to both the service staff and the sales force. In just about every organization we've spoken with, sales, marketing, and customer service divisions qualify as the best types of voice mail clients we've seen.

With one call a regional marketing director can send important, time-critical information to the entire sales staff. Hundreds upon hundreds of individuals, all away from their phones, get the same message. Their customers can leave orders through voice mail. Numerous companies in the auto parts, computer systems, and catalogue order entry industries take orders by the thousands through voice mail. And they all have one thing in common: They add value to what they do without giving up anything.

For direct-response marketing campaigns, voice mail increases the capture rate. Voice mail services are being streamlined for specialized uses. The needs of virtually every conceivable approach can be served effectively. We have witnessed tremendous acceptance of direct-response types of applications for direct mail, print media advertisements, and classified ads. All of the callers are nonsubscribers, effectively using voice mail—by the thousands.

One regional airline extensively analyzed customer acceptance after an introductory program and found that more than 90% of those who used voice mail liked it and would use it regularly. And the results were predictable. The service was convenient, and timely information could be delivered flawlessly.

In a similar type of application, financial advisors, stockbrokers, and portfolio managers can give their customers information at less cost, with better results, and with more accuracy than ever before. Portfolio status reports, buy and sell orders, and trade confirmations can be sent and received through voice mail. Many national brokerage and trading organizations use voice mail to give better service to their customers.

Accounting firms and management consultants use voice mail for both internal project status reporting and communicating with their clients. In an environment where everyone is away from the telephone, voice mail is indispensable.

Internal communication within large organizations has many behavioral dynamics attached. Voice mail solves the problems of real-time confrontation or intimidation and the problem of telling someone bad news directly. The end user now has a way to tell someone something without having to do it directly; and it gives everybody the option of thinking about the message until a time when they are better prepared to respond.

The costs associated with telephone tag are highly overstated. Voice mail does indeed save time spent making unsuccessful direct calls. This can lead to substantial savings in the cost of long-distance calls, but the net effect only offsets the cost of using voice mail. The real benefit voice mail can provide in the telephone-tag situation is that it can lead to revenue opportunities and to a better competitive position.

At Voicemail International we have developed extensive analytical capabilities and sophisticated econometric models of the true costs of telephone tag. Our clients struggled to find ways to come up with cost/benefit figures to justify the use of voice mail. Predictions were made that voice mail could reduce the cost of telephone tag from 60% to 90% in toll and time-spent equivalents. The results after voice mail was fully implemented are important to note:

We found that for every call placed through voice mail, time and toll-charge savings were indeed impressive, often exceeding predicted savings. However, overall savings did not filter down to the bottom line. Operating costs commingled too many other variables to prove any measurable benefits, except in one case—revenue growth showed a measurable increase. In this case, the oftener voice mail was used, the more time could be spent being productive. People using voice mail now had more time to spend—time they soon began using effectively to add value to what they did. The phone bill, in several cases, actually increased after making the decision to use voice mail; but the cost of adding voice mail was offset by hard dollar savings, and the real benefits came in the form of better productivity and added convenience. These factors are virtually impossible to quantify and measure on a short-term basis. In the long term, when voice mail has been effectively implemented and the right people are using it, it really works.

An important element in a voice mail system is the flexibility it has for serving both subscribers and nonsubscribers. More than 60% of the telephones in this country do not have touch-tone-generating capability. Portable touch-tone generators are virtually useless from pay phones and most hotels because of the poor quality of phone microphones. We provide 24-hour-a-day operator assistance for callers to sign on and use the various features. This live intervention maximizes the results, especially the first time a nonsubscriber calls.

Earlier it was mentioned that economic realities are going to drive decision makers to implement voice mail programs within their organizations. The decision maker today looks at cost reduction; but what companies will get with voice mail is revenue opportunities. Voice mail is a supplementary, or alternative, method of communication—a whole new medium that is just beginning to be understood.

Security of message handling, simplicity of operation, and user-friendly characteristics tell us that the technology is functionally capable of serving many diversified needs. What is developing now is the understanding that the entire global population must have access. Voicemail International is on that track, to link the entire international public switch so that it can be voice-mail-capable.

Another key factor to open up the world of voice mail is the ability to be open-ended. This means that two nonsubscribers can communicate by voice mail with each other. Toll-free access to the caller, and the ability to charge it to a variety of bank or travel cards, gives the necessary flexibility. Delivery of the voice message to any person at any telephone on a deferred basis gives the service people need. And it doesn't have to be a one-to-one type of call; one nonsubscriber can send a message to any number of people.

As advocates of this technology, our goal is to network the entire global telephone system with voice mail services. Voice mail will mobilize the world's information resources. By 1990, it will be a household word.

Electronic mail: Evolving from intracompany to intercompany

by H. PARIS BURSTYN
The Yankee Group
Boston, Massachusetts

ABSTRACT

The roots of electronic mail technology go back to the first facsimile systems, but most people today think of electronic mail as computer-based message systems (CBMS)—either remote electronic mail services or electronic mail software packages running on in-house computers. Systems and services providing access to databases and data processing as well as simple command and menu structures will attract new users. Over the next five years, as electronic mail use grows, the people who communicate within companies (intracorporate communications) will want to contact people outside their company (intercorporate communications). Value-added services that provide private messaging systems must begin providing access between organizations' private networks and the value-added networks. They must also make possible user-transparent access between organizations that subscribe to their services. Today only a few services provide this facility, and then only by special arrangement. Eventually it will not matter which network a user signs onto to receive mail; gateways will provide transparent access between the networks to make it possible to have internetwork and international electronic mail.

INTRODUCTION

Electronic mail technology traces its roots to the first facsimile systems. But most people today think of electronic mail as computer-based message systems (CBMS). These systems stem from the first remote timesharing service bureaus and have evolved into software packages for in-house computers. Users can now meet their messaging needs by choosing remote electronic mail services, electronic mail software packages that run on their computers or turnkey systems.

In the early days of data processing, hardware vendors offered users little in the way of applications software. The third-party software industry evolved to address that market. Some companies provided software for customer-owned hardware; others offered to run programs on their computers—either interactively from terminals at the user site or as batch jobs. To compete with this new industry, hardware vendors began offering more sophisticated programs. The timesharing companies countered with additional programs. Among these programs were electronic mail systems.

With the advent of electronic mail, timesharing companies held an initial advantage over hardware vendors: They were communications-oriented from the start. In many cases they had to interconnect multiple sites. Although hardware vendors also sold to multiple sites, they did not connect them.

The first timeshared messaging systems—Comet, from the Computer Corporation of America (Cambridge, Massachusetts) and OnTyme from Tymshare (Cupertino, California)—and their followers aimed at meeting the over 80% of business communications between people in the same company. The systems allow messaging between all those at the corporation who have access to the service. These services help users escape telephone tag and USPS delays by providing a simple means to send short (4- to 6-line) messages.

Recognizing that in-house computers could more effectively, in cost and usage, meet intracorporate communications needs than could outside services, computer vendors and some software houses developed electronic mail programs for installed hardware. To send a message within a single facility, it makes more sense to take advantage of the on-site computer than it does to dial up a remote computer, which involves additional communications and timesharing costs. Employing an in-house computer for electronic mail allows users to add an application to help justify the cost of the computer and its peripherals.

Recently, hardware manufacturers began offering software programs that run on the computers they sell to end users. Third-party software houses also wrote programs for specific hardware located at customer sites.

Now virtually every major hardware and software vendor and value-added carrier and a number of timesharing companies offer electronic mail packages.

TRENDS IN ELECTRONIC MAIL

During the next five years, electronic mail use will skyrocket. As it does, the people who communicate within companies will want to contact outside people and therefore will want to bring the conveniences of intracorporate electronic mail to intercorporate communications. Intercorporate links can speed order processing, improve inventory control, and establish direct-payment channels for various transactions. This usage growth (See Table I) will spark the development of bridges and gateways between services and in-house software.

Market growth projections show that the number of service-bureau-resident mailboxes will double each year through the end of 1984 and that growth will then slow to 50% each year during 1985 and 1986. Private-system mailboxes will increase by 50% each year through 1984 and then by 75% in both 1985 and 1986.

Concurrent with this spectacular growth, the market will see a major shift in use and applications. The 200,000 service-based mailboxes in use during 1983 will see heavy intracorporate messaging applications, and internal mail systems will experience a mixture of messaging and management support applications (like calendars and reminders).

After 1983 there will be a change in the way that systems and services are applied. Private electronic mail systems will pick up the bulk of intracorporate messaging, and public services will provide intercorporate messaging. Public and private telephone systems provide a good comparison. Corporations install tieline networks for intracorporate communications and use the public-switched telephone network (PSTN) for outside communications. Employees receive two telephone numbers, one for the tieline-based system and one for the PSTN. Similar communications systems will evolve for

TABLE I—CBMS mailbox growth projections

Year	Service Users	Private Users
1981	49,000	80,000
1982	98,000	100,000
1983	196,000	150,000
1984	392,000	225,000
1985	588,000	393,750
1986	882,000	690,000

Source: The Yankee Group.

electronic mail. Employees will have IDs on their company's internal mail system as well as on at least one public system. They will use the internal system for intracorporate mail and the external system for intercorporate mail. They might require multiple public mailboxes to accommodate correspondents who have mailboxes on different systems.

With this evolution will come growth in mailbox numbers for the services, but a drop in traffic, as users turn to their internal systems for the majority of their communications. To counter this trend and to make up for the fact that they address only 20% of business communications applications, not 80%, public electronic mail services must provide additional capabilities. The Yankee Group expects the electronic mail services to start with public databases (as some already have), like news wires and stock quotations. Later, they will provide communications links for specific applications between user groups on their services. These applications will include customer/supplier communications like invoices, purchase orders, and electronic funds transfers. Finally, intersystem personal communications will take place. Today some of the public electronic mail services allow intercorporate communications; in most cases it requires making special arrangements between the two companies and the service provider.

Historically, electronic mail services have not been very user-friendly. Even the most recent market entries deliver only minor improvements in the user-service interface. All the services require cumbersome sign-on procedures, and message editing requires the skills of a safecracker.

Since electronic mail is usually aimed at managers and professionals, these difficulties combine to keep usage low. Since it is a new technology, getting people to try it is half the battle. Because the services are difficult to use, vendors have frequently lost the battle.

Increasing market pressure will change this situation, because it will contribute to the development of new features and capabilities by the services. Under the influence of recently introduced electronic mail packages—such as Bolt, Beranek, and Newman's InfoMail; Digital Equipment's DEC-mail; and IBM's Professional Office System (PROFS)—services such as Comet, Dialcom, InfoPlex, and Telemail are extending their capabilities and becoming easier to use.

As electronic mail systems gain a wider range of capabilities—database and data processing access as well as simple command and menu structures—more users will find the various alternatives increasingly attractive. Concomitant with an increase in users comes increased use and demand for more features. Further, during the last year sufficient market pressure developed to drive suppliers to provide more functional systems.

Electronic mail no longer refers to simple messaging capabilities. Today's systems deliver sophisticated mailing features like message forwarding, note attachments (buck slips), multiple copies, and mailing lists. Additionally, the systems provide management support functions, like word processing, hierarchical filing, calendar management, tickler files, and reminders. The services are just beginning to provide capabilities like calendar services and access to public databases.

Although the services remain the functionally simplest of electronic mail media, constant pressure from software vendors will force them to provide comparable functions. Increased user-friendly functionality will promote higher usage levels, especially when systems improve management efficiency, elminate telephone tag, and speed effective communications.

As usage increases, intracompany electronic mail users will want to bring their electronics-based benefits to intercorporate messaging—benefits like bill paying, order entry, and inventory updates. Just as people today know they can reach almost anyone through paper-based mail services, people using electronic mail systems will eventually want to connect individual systems so that they can exchange intercorporate as well as intracorporate mail. Therefore, the value-added services that provide private messaging systems will begin providing access between organizations' private networks and the value-added networks (VANs) as well as between organizations subscribing to their services (this can be specially arranged today on some services). The step following that will be gateways between networks, which will overcome today's networking Tower of Babel.

It will not matter which network a user signs onto to receive mail. Gateways will provide transparent access between the networks to allow internetwork electronic mail. Today it is hard to imagine representatives of the different VANs getting together to discuss gateways between their networks, but the business pressures that will force such meetings are already evolving.

The IBM Information Network and American Bell's AIS/Net 1000 aim specifically at intercorporate communications. And American Bell goes so far as to state a desire to be a "network of networks."

Last October IBM's Information Network won a major contract from the Insurance Institute for Research to provide a value-added network that will allow property and casualty insurance companies to communicate with their independent insurance agents. This Insurance Value Added Network Services (IVANS) will deliver communications capabilities between some 70 different computers and terminals, including many non-IBM devices.

IVANS will provide a range of communication processing functions beginning in mid-1983:

1. Store-and-forward message switching, both prescheduled and on demand
2. Interactive access to network-resident application programming
3. Access to an insurance industry database
4. Speed, protocol, and data format conversions to allow dissimilar terminals to communicate
5. Storage of messages and data
6. Support for user-written programs
7. Network security, management, and documentation

IVANS is an ambitious project for bringing an entire industry on line. IBM has combined communication processing with timeshared application processing and has also offered the network on an intercompany basis. Many other such industry networks will be implemented in the near future for trucking/transportation companies, airline and hotel reservations, banking, retailing, and similar fields. Vendors competing to provide these quasi-private networks include not only IBM and American Bell, but hardware vendors like Tandem and Control Data, service bureaus like Tymshare and NCSS, and carriers like Isacomm and SBS.

The forerunner of these systems, the Transportation Data Coordinating Committee, has developed standards for electronically communicating purchase orders and invoices for the trucking industry. The grocery industry has also established a data communications standard (the Uniform Communication Standard) for orders and shipments between major food processors/manufacturers and large warehousing/wholesale organizations.

To further international and intersystem communication, the CCITT formed a study group to develop international standard protocols for connecting existing and future CBMS and document distribution systems. It is called the CCITT Special Rapporteur Group on Message-Handling, Study Group VII.

Among recent decisions, this group drafted access protocols to enable users of teletex, facsimile, and related systems to make use of CBMS-based message transfer services. It also refined and extended existing specifications for the relay protocol governing message exchange between separate domestic and/or foreign CBMS systems. This specification now includes the control information that acts as an electronic "envelope" for messages in transit.

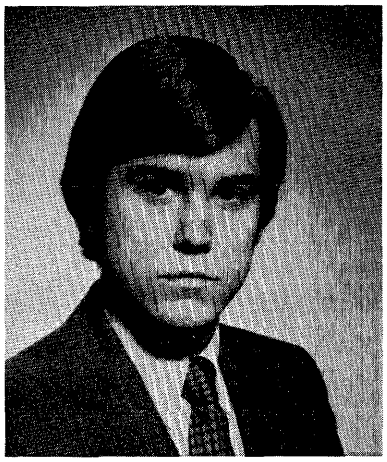
These CCITT steps follow the recently-agreed-upon National Bureau of Standards Specification for Message Format for Computer Based Message Systems. Bolt, Beranek, and Newman; Computer Corporation of America; Dialcom Inc.; Digital Equipment Corp.; GTE Telenet Communications Corp.; and Tymnet have all agreed to support that standard. Other electronic mail service and system providers have said they will back the standard if enough demand develops.

IBM has seriously considered document communications integrity between its systems. Its philosophy takes shape in the Document Interchange Architecture, Document Composition Architecture, and Graphic Codepoint Definitions. IBM supports these blueprints on the 5520 administrative system, the Displaywriter word processor, and the 8100 computer under the Distributed Office Support Facility (DOSF) and the Distributed Office Support Systems (DISOSS) software programs. Recently these architectures were spread to the majority of IBM's office systems, most notably the Professional Office System (PROFS).

Outside IBM, standards currently agreed upon and those under discussion cover only message addressing; message format integrity remains to be formalized. Current standards like HDLC and X.25 do not guarantee that a message formatted and sent from one system will appear the same way at the recipient's terminal. This issue has just started to be addressed. In 1984 the CCITT is scheduled to formalize the Group 4 facsimile standard. Group 4 will be teletex compatible, and the two systems will allow text and graphics to be transferred between systems with full integrity.

Increasing system/service sophistication and compatibility will lead to new applications and to growing numbers of users. These developments augur the advanced, interconnected electronic mail systems that will evolve from today's individual systems and services.

DECISION SUPPORT SYSTEMS


Todd Ziesing
Ross Systems, Inc.
New York, New York

The series of sessions on Decision Support Systems (DSS) tackle the questions that DSS faces in the 1980s. How does a manager today effectively use the technology of mainframe, minicomputer, and microcomputer to produce cost-effective solutions for decision support? Several sessions directly address this question and come to some very different conclusions.

1. The first proposes that "Micros Can Do It All."
2. The second tells convincingly why only *custom,* micro-based decision support systems can address this problem.
3. Two sessions cover all aspects of designing systems of this kind.
4. A fifth session presents micros, minis, and large computers in a distributed financial planning/decision support environment.
5. A final session discusses how these technologies are all being integrated into executive information systems for senior corporate executives.

These sessions meet the most puzzling questions of the micro versus mainframe debate head on. There is substantial material here for both the management information system user and the non-computer-oriented decision maker in any corporation.

A new look at existence dependency in databases

by T. C. CHIANG*
American Bell Inc.
Piscataway, New Jersey

ABSTRACT

To ensure data consistency, existence dependencies between records must be preserved when the records are updated. It is necessary to identify such dependencies when the system is designed, so that update operations can be performed correctly.

This paper presents a new look at existence dependencies in databases. It identifies a set of basic update rules that can be incorporated into a database management system (DBMS) to preserve existence dependencies between records. The paper shows that the existence dependencies supported by other existing commercially available DBMSs can be defined precisely in terms of the basic rules. Furthermore, a combination of these basic rules captures new existence dependency semantics that are not handled by the commercially available systems. This paper also discusses solutions to the "interference" problems associated with closely related records. These interference problems have never been discussed in the literature before, but are important to system operations.

*This paper was written while T. C. Chiang was working at Bell Laboratories, Whippany, New Jersey.

A. INTRODUCTION

In an enterprise that has complicated relationships among entities, the existence of one entity often depends on that of others. Since a database represents a real world, existence dependencies in that world are reflected on the database as existence dependencies among data records. For an application that requires frequent deletion and insertion of records, existence dependency in the database must be handled correctly to avoid data inconsistency.

A reasonable place in a database system to handle existence dependency is in the database management system (DBMS) (as opposed to in the user's programs). Most of the commercial DBMSs such as IBM/IMS or other CODASYL-based systems handle a few kinds of existence dependency. For example, in IMS, a deletion of a root segment instance will trigger the deletions of all child segments instances under the root segment. In CODASYL based systems, there are AUTOMATIC and MANUAL, and MANDATORY and OPTIONAL declarations for memberships in a CODASYL set, which define certain kinds of dependency among records. For example, if a member is inserted into a set, it cannot be deleted independent of the set. These mechanisms are not rich enough to capture other kinds of existence dependency in most databases.

In the research community, little attention has been given to the theory of existence dependency. For example, in the early work on relational data model, much attention was given to functional dependency and normalization. The term *update anomaly* was used to define all the problems related to existence dependency. However, in reality, many of the cases of anomaly are not really anomalous but are expected existence dependency and should be handled as such. In the recent years, some attention has been paid to the problems of existence dependency.[1–5, 7–8] Chen defined the concepts of regular and weak entities to capture some semantics of existence dependency among entities.[1] In a later paper, Dogac and Chen mentioned the concepts of update propagation.[2] Similarly, Smith and Smith talked about the ideas of triggered updates that involve automatical updating of dependent records.[3] Keller and Wiederhold listed a number of existence dependencies that were considered important in keeping the database consistent.[4] None of these papers treated the problems of existence dependency extensively enough to capture most of the dependency semantics. Chiang and Bergeron described a system that handles a set of existence dependencies.[5] However, no detail on other problems related to existence dependency has been presented. Navathe and Schkolnick[7] talked about update rules in the framework of view representation, and Date defined a rich language to describe existence dependencies as referential integrity for the relational data model.[8] However, neither of these papers broke down the existence dependency semantics into atomic units or mentioned the interference problems.

In this paper, existence dependency and its related problems are intensively discussed. Existence dependency is viewed as a property of a relationship among entities. A *coupling factor* is defined as a set of update rules for handling existence dependencies between entities. This paper shows that update rules supported by the commercial systems (e.g., IMS and CODASYL systems) and by others can be described in terms of the coupling factors.[2, 4] Furthermore, the new look of the existence dependency problems enables us to discover new dependency semantics. A DBMS will enforce the rules for handling existence dependency, so that record deletions and insertions will not turn the database into an inconsistent state. Problems arise when two relationships exist within the same set of entities and the existence dependencies of the relationships interfere with each other. Later sections of this paper will define the sets of various kinds of existence dependency and of their interferences, and will present an algorithm to detect the interferences.

The result described in this paper is actually implemented in a DBMS for telephone business applications in the Bell telephone companies over the United States.[5]

B. EXISTENCE DEPENDENCY

An extended entity-relationship (E-R) data model is used as a basis for dealing with existence dependency. Roughly speaking, a basic E-R model views a database consisting of files that are sets of records and relationships among records.[1] In the extended E-R model, a set of relationships is referred to as an association.[5] An association has a *coupling factor* as one of its properties. A coupling factor is a set of existence dependencies. To simplify the representation, only the existence dependencies of binary associations are discussed in the rest of this paper. Therefore, the term *association* will mean binary association from now on. Also, an association will be viewed as a binary relation with the two files involved as its domains.

Although there may exist many kinds of existence dependency, four basic ones are identified; the definitions are given as follows. Let $E1$ and $E2$ be two files, and A an association between $E1$ and $E2$. Let $e1$ be any record in $E1$ and $e2$ a record in $E2$ type associated with $e1$ via A; then $e1$ and $e2$ are referred to as A-associated. The basic existence dependencies can be defined as the following update rules:

1. An $e2$ cannot be inserted unless there already exists an

A-associated $e1$. An insertion of $e2$ implies establishing a relationship between $e1$ and $e2$.
2. The deletion of an $e1$ implies the deletions of all A-associated $e2$s.
3. An $e2$ cannot be deleted, if there exist an A-associated $e1$.
4. A relationship cannot be deleted, unless $e1$ or $e2$ is deleted.

Note that normally deletion of $e1$ or $e2$ implies the deletion of the relationship between $e1$ and $e2$. The insertion of a relationship is allowed only if both $e1$ and $e2$ exist. Also, the dependencies specified by the rules are directional and transitive. For example, in the descriptions of the rules above, $E2$ depends on $E1$. Furthermore, if $E2$ depends on $E1$ and $E1$ depends on $E3$, it implies that $E2$ depends on $E3$.

A set of coupling factors can be defined as the power set of the basic rules. Thus, there are 16 possible coupling factors, including the null coupling factor that has none of the above dependencies. By Chiang and Bergeron, a "very tight" coupling factor is defined as one that has all four dependencies.[5] The other coupling factors represent various kinds of dependencies between records. The coupling factors are unidirectional, that is, all the rules of the coupling factor have the same direction, which is defined as the direction of the coupling factor. Coupling factors are also transitive. The transitive property of coupling factors is the key to the propagation of updates.

With these definitions of coupling factor, we can show how update rules supported by others can be described in terms of coupling factors.

B.1 IMS

The update rules for IMS are defined by its hierarchical structure. Therefore, the deletion of one node in the hierarchy implies the deletion of all its child nodes. An insertion of a node into the hierarchy requires the existence of its parent node.

The IMS update rules can be viewed as the coupling factor consisting of rule 1, rule 2, and rule 4 above. For example, consider a subset of the education database presented by Date.[6] Figure 1 shows the structure for the educational database.

A student segment can be inserted only if there is an offering of a course (i.e., rule 1), and the deletion of an offering of a course implies the deletion of all student segments associated with the offering (i.e., rule 2). Furthermore, IMS does not allow the breaking up of hierarchical relationships (e.g., rule 4). Note that IMS does not support rule 3.

B.2 CODASYL Systems

In CODASYL systems, membership class in a set,[6] is used to describe the update rules. Let Owner (O) and Member (M) be two record types, and OM be the set defined between O and M. The membership for m of M in OM is said to be

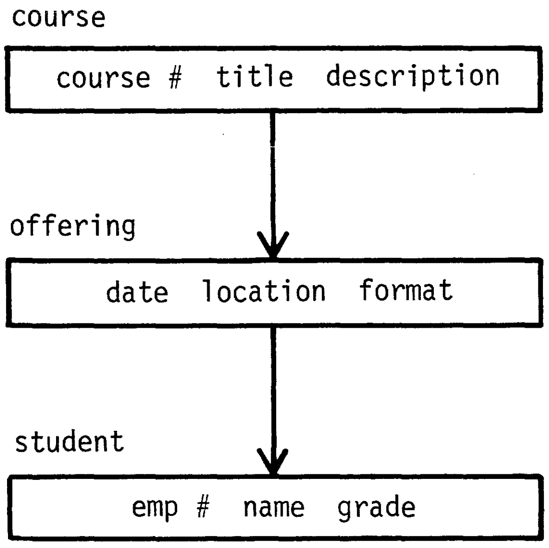


Figure 1—Education database structure

mandatory, if m cannot be removed from the set and deletion of an owner o implies the deletion of all members. If the membership is *optional,* m can be removed from OM without deleting m from the database. If the membership of M in OM is *automatic,* the DBMS will automatically establish the membership in OM when an m of M is inserted. If the membership is *manual,* then the programmer has to establish the membership explicitly. There are four kinds of memberships: (1) mandatory-automatic, (2) optional-automatic, (3) mandatory-manual, and (4) optional-manual. We can use coupling factors to describe these memberships precisely. Mandatory-automatic is the coupling factor containing rules 2, 4, and 1; optional-automatic is the coupling factor containing rules 2 and 1; mandatory-manual is the coupling factor containing rules 2 and 4; and optional-manual is rule 2.

B.3 Relational Systems

In relational systems, there are no comparable update rules. Existence dependency is treated as anomaly in the context of normalization. For example, consider a supplier relation:

S′ (S#, SNAME, STATUS, CITY)

S′ is in 2nd normal form,[6] because all non-key attributes depend on the key and STATUS depends on CITY. This implies that information about STATUS cannot be inserted until some supplier in the city is in the database, and deletion of the only supplier in a CITY will delete all STATUS and CITY information. These are considered to be anomalies by Date.[6] However, they may be valid existence dependencies for some applications. The coupling factor in this case contains rule 1 and the rule to be described in Section D.

To show the problems in dealing with existence dependency in a relational system, let us consider another example. Consider a database consisting of three relations: (1) suppliers (SUPP), (2) parts (PART), and (3) shipments (SHIP). Each of the relations has a set of attributes and a set of tuples as follows:

SUPP (S#,	SNAME,	STATUS,	CITY)
s1	adams	20	Chicago
s2	baker	10	Newark
s3	chang	30	L.A.
•	•	•	•
•	•	•	•
•	•	•	•

PART (P#,	PNAME,	COLOR)
p1	chip	red
p2	LED	green
p3	fan	red
•	•	•
•	•	•
•	•	•

SHIP (S#,	P#,	QTY)
s1	p1	10
s1	p3	5
s2	p2	20
s2	p3	8
•	•	•
•	•	•
•	•	•

Suppose we want to delete all red parts from the database (from both PART and SHIP). Using a pseudorelational language, we can express the deletion operations as follows:

1. delete SHIP tuples, where SHIP.p# = PART.p#, and
2. delete PART tuples, where PART.color = "red".

Now, if we reversed the order to do step 2 first, then step 1, the deletion of the SHIP tuples would not be possible, because the information about red parts would not be available at that point. Currently, there is no relational system to prevent deletions in a wrong order. If we consider SHIP as an association between PART and SUPP, then the deletion of red parts will trigger the deletions of the SHIP relationship tuples.

We have shown that the idea of coupling factor can be used to describe the existence dependency supported by some existing systems and to capture other dependency semantics that are not recognized by the existing systems. Coupling factors have many other interesting properties; one of them will be discussed in the next section. The problems discussed in the next section have never been handled by the existing systems.

C. INTERFERENCES

An interference occurs when two adjacent associations have coupling factors that interfere with each other. For example, suppose that we have a database consisting of a department file, an employee file, and a project file, and that the update rules for the data base are as follows:

1. An employee record cannot be inserted, if there is no department that the employee can work for
2. A project record cannot be inserted if there is no employee in department that can handle the project
3. A department record cannot be inserted, if there is no project for it.

If these update rules are enforced, then no record can be inserted into the database. In a system that supports more complex relationships between records, the interference between two coupling factors becomes a problem. In this section, we shall talk about this kind of interference. We shall use the first three coupling factors that were presented in Section B to illustrate the problem. More detailed definitions are given as follows.

Adjacency.—An association A is said to be adjacent to another association B, and vice versa, if and only if A and B have a domain in common.

Parallel adjacency.—Associations A and B are said to be parallel adjacent if and only if they have the same set of two perhaps distinct domains.

Serial adjacency.—Associations A and B are said to be serial adjacent if and only if A and B have exactly one domain in common.

Interference state.—Two adjacent associations are said to be in an interference state if and only if no record can be inserted in or deleted from any of their domains.

Interfering coupling factors.—Two coupling factors are said to be interfering with each other if and only if assigning them to two adjacent associations would cause an interference state.

Now we are ready to look at all possible interference states generated by pairings of the eight possible coupling factors.

C.1 Parallel Interferences

Parallel interference occurs when two parallel adjacent associations have interfering coupling factors. It can occur only when two coupling factors have opposite directions. Therefore, all theorems in this section assume that the coupling factors are in opposite directions. *e1* and *e2* are used to represent records in the two domains, E1 and E2, of an association. The rules are to be assigned to two distinct associations.

Theorem 1. Rule 1 interferes with rule 1.

Proof. Since the insertion of $e1$ depends on the existence of $e2$, and vice-versa, the associations would be in an interference state.

Theorem 2. Rule 3 interferes with rule 3.

Proof. Since $e1$ cannot be deleted without first deleting $e2$, and vice-versa, the association would be in an interference state.

Clearly two coupling factors are interfering with each other if they contain one interfering rule.

To identify all the possible interference states, we use three Boolean variables, *X, Y,* and *Z,* to represent the presence of rules 3, 2, and 1, respectively, in a coupling factor. A Boolean variable has the value 1, if a rule is present. Thus, the eight coupling factors can be represented as

XYZ
000
001
010
011
100
101
110
111

A matrix that represents all the possible states is shown in Figure 2, where an x indicates an interference state. A null coupling factor, which contains none of the rules, does not interfere with any other coupling factors. From Figure 2, it can be seen that there are in total 28 interference states, which are easy to detect: one performs an intersection on the coupling factors involved; if the resulting set contains either rule 1 or rule 3, there is an interference. By considering each coupling factor to be represented by three Boolean variables, one can even set up a hardware machine to realize the Boolean function represented by the matrix in Figure 2.

C.2 Circuit Interference

The concept of parallel interference can be generalized to produce concepts of other interferences. Defining an E-R diagram with respect to existence dependency as a labeled directed graph, we use nodes to represent files and edges to represent associations. A label on an edge represents the association name (which uniquely identifies the edge), and the coupling factor. Figure 3 shows an example of an E-R diagram conveying dependency information, where E1 and E2 are nodes representing files, A is an association, and C is the coupling factor of A. The arrow on an edge represents the direction of the coupling factor. Since the arrow is pointing from E1 to E2, E1 depends on E2 via A/C. E1 is referred to as the initial node and E2 the final node.

	000	001	010	011	100	101	110	111
000								
001		x		x		x		x
010								
011		x		x		x		x
100					x	x	x	x
101		x		x	x	x	x	x
110					x	x	x	x
111		x		x	x	x	x	x

Figure 2—Interference states

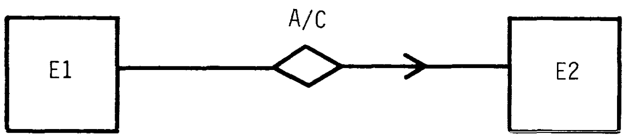


Figure 3—E-R diagram with dependency

A path in an E-R diagram is a sequence of serial adjacent edges, where the final node of one edge is the initial node of another. Figure 4 shows an example of a path, where A1, A2, . . . , An are names of the associations. A circuit is a path A1, A2, . . . , An in which the initial node of A1 is the final node of An.

A circuit interference is defined as an interference state caused by the interfering coupling factors of a circuit of associations. With the algorithm in Section 1 and the transitive property of the dependency rules, we can reduce the problems of circuit interference to those of parallel interference. For example, Figure 5 shows a circuit interference.

To detect such a state, one could

1. Detect a circuit.
2. Starting at a node of the circuit, reduce two serial adjacent edges and the common node to a "virtual" edge, *V,* with a coupling factor that is the intersection of the two original associations.
3. Repeat step 2 until two parallel adjacent associations result.
4. Check parallel interference using the algorithm in Section C.1.

D. FUTURE WORK

There are many interesting problems related to existence dependency.

1. To identify more existence dependence rules. For example, a variation of rule 2 can be stated as follows: The deletion of an *e*1 implies the deletion of all A-associated *e*2s, *if el is the only A-associated el of the e2s.* This rule captures new semantics of dependency and will generate more interference states when it is paired with other rules.
2. To extend the definitions of coupling factors to *n*-nary associations.

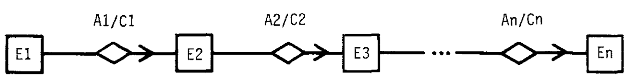


Figure 4—Example of a path

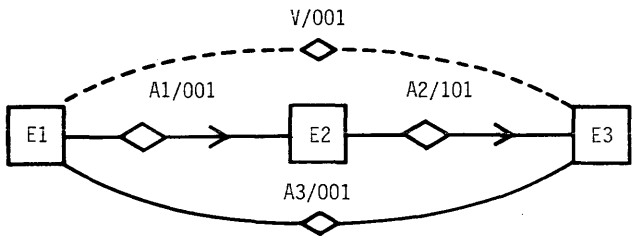


Figure 5—Circuit interference

3. To apply the concept of coupling factor to update dependency in general. Existence dependency is one kind of update dependency. The other kind is *modification dependency*. Modification dependency occurs when the modification of a data item depends on the modification or existence of other data items. For example, a deletion of an employee record may trigger the modification of the total number of employees in a company.

D. CONCLUSION

We have considered existence dependency as a set of data semantics that has the same importance as that of functional dependency has in relational theory. We have described the problems of existence dependency in databases. We have identified a set of existence dependency rules from which a coupling factor of an association can be defined. We have described some interesting properties of the coupling factors and identified many interference states and the algorithms to detect them. We believe that there are still many interesting problems in existence dependency left to be discovered.

REFERENCES

1. Chen, P.P. "The Entity-Relationship Model—Towards a Unified View of Data." *ACM Transactions on Database Systems,* 1 (1976), pp. 9–36.
2. Dogac, A., and P.P. Chen. "Entity-Relationship Model in the ANSI/SPARC Framework." *Proceedings of the Second International Conference on Entity-Relationship Approach* (October 1981), pp. 361–378.
3. Smith, J.M., and D.C.P. Smith. "Database Abstractions: Aggregation and Generalization." *ACM Transactions on Database Systems, 2* (1977), pp. 105–133.
4. Keller, A.M., and G. Wiederhold. "Validation of Updates Against the Structure Database Model." *Proceedings of the Symposium on Reliability in Distributed Software and Database Systems* (July 1981), pp. 195–199.
5. Chiang, T.C., and R.F. Bergeron. "A Data Base Management System with an E-R Conceptual Model." *Proceedings of the First International Conference on the Entity-Relationship Approach* (December 1979), pp. 467–476.
6. Date, C.J. *An Introduction to Data Base Systems,* Reading, Mass.: Addison-Wesley, 1976.
7. Navathe, Shamkant B., and Mario Schkolnick. "View Representation in Logical Database Design." *Proceedings of the SIGMOD Conference* (May 1976), pp. 144–156.
8. Date, C.J. "Referential Integrity." *Proceedings of the VLDB Conference* (October 1981), pp. 2–12.

Issues in the design of relational model management systems

by ROBERT W. BLANNING
Vanderbilt University
Nashville, Tennessee

ABSTRACT

One component of the literature on model management in decision support systems (DSSs) suggests that the relational framework for data management be extended to the management of decision models. This paper examines three important design issues in relational model management: the organization of relational model banks, relational completeness of model query languages, and system implementation.

INTRODUCTION

The growing use of decision models in decision support systems (DSSs)[1,14,15,22,30] and the successful implementation of database management systems (DBMSs) have given rise to suggestions that (1) decision models, like data, are an important information resource that should be managed effectively, (2) a discipline of model management be developed to help DSS users to make more effective use of models, and (3) model management systems be developed to implement important model management concepts.[17,18,24,25,30,31,33] Examples of such models are mathematical programming models for production and distribution scheduling, Monte Carlo simulations of transportation networks, and deterministic simulations of the financial structure of an enterprise. Although no model management systems now exist, it can be argued that the systems that support such financial planning languages as EMPIRE[13] and IFPS[19,23] are forerunners of the model management systems that will eventually be implemented, just as the file processing systems of the 1960s were forerunners of DBMSs.

One of the frameworks proposed for model management is a relational framework: it is suggested that a model bank may be viewed as a set of (virtual) relations with input and output attributes and functional dependencies between them, just as a database may be viewed as a set of relations with key and content attributes.[4,5,6,7,8,10] Two advantages are offered for such a perspective. First, since some of the operations performed with models have counterparts in relational data management, important concepts in the latter area may be useful in the former. For example, the execution of a model is similar to selection or restriction of a data relation. Similarly, whenever several models are used to respond to a query such that the outputs of some models are inputs to others, an operation similar to a join is performed. Clearly there are substantial differences between relational data and model management at the level of system implementation and program execution, but we are concerned here only with the user view of models, not with the procedures for model execution. The second possible advantage is that it may eventually be possible to synthesize the relational frameworks for data and model management to produce a single relational framework for information management in DSSs.[8]

Three principal issues have been identified for relational model management:

1. *Model bank organization.* Should a model bank be viewed by a DSS user as a single "universal" model or as a set of separate models? If the latter, what are the criteria for partitioning the universal model? Are there normal forms for relational model banks as there are for relational databases?
2. *Relational completeness of model query languages.* What operations should be specified by a query language for relational model management? How do they differ from those of relational data management? What are the linguistic properties (e.g., position on the Chomsky hierarchy) of a relationally complete model query language?
3. *Implementation issues.* What are the issues with regard to security, integrity, query optimization, and the implementation of joins in relational model management? With regard to the latter, if an output of one model is an input to another and vice versa, how is a consistent solution for the model bank to be found?

These three issues will be examined in the following three sections, after which we will discuss the broader issue of model management in DSSs.

MODEL BANK ORGANIZATION

The question of whether several interrelated operations (such as preparing a sales forecast and then simulating the impact of the resulting sales volume on production processes) should be performed by one model or by several models has been raised in the model management literature. It has been suggested that models be decomposed by organizational function (finance, marketing, etc.),[33] organizational level (the strategic, tactical, and operational levels),[29] model solution technique (simulation, linear programming, etc.),[3,9] or a combination of these.[12]

Another approach, similar to that taken in relational data management, consists of identifying anomalies that can arise in using models and then deriving normal forms that result in the elimination of these anomalies.[6] The anomalies of interest in relational data management are update anomalies—problems that can arise in adding, deleting, or changing tuples in a relation. Such anomalies do not arise in relational model management, because the tuples in a model relation do not exist in stored form; rather, they are generated as needed in response to user queries. However, other anomalies, called processing anomalies, are found in model management, and they give rise to normal forms similar to, but not identical with, those of relational data management.

There are three types of processing anomaly, which lead to three types of normal forms, called the alpha, beta, and gamma forms. These anomalies are

1. *Input anomalies.*—An input anomaly occurs whenever a user, in order to require that an output of a model be

calculated, must enter at least one input that is not needed for the calculation of that output. Input anomalies are similar to (but not identical with) the anomalies that lead to the second normal form in relational data management.

2. *Search anomalies.*—A search anomaly occurs whenever there is a transitive dependency in a relation, for example of the form {price⇒sales volume, sales volume⇒production cost}; this requires that the user who wishes to determine the production cost resulting from a given sales volume enter different values of the price until that volume is realized, and with it the corresponding production cost. The anomaly is eliminated by projecting the relation into the two relations {price⇒sales volume} and {sales volume⇒production cost}; this projection is equivalent to the third normal form in relational data management.
3. *Output anomalies.*—An output anomaly is a nondeterministic response to a user query. One cause of nondeterminism is the presence of two or more identical output attributes in different models (e.g., in two models, both of which calculate production cost). The elimination of such anomalies leads to a normal form in which the output attributes of all of the models are pairwise disjoint. Such a normal form could not be implemented in relational data management, because some output attributes, such as a city name, can be associated with suppliers, customers, regional offices, and so on. Another cause of nondeterminism, which cannot be resolved by projection into a normal form, arises during the implementation of joins. This is explained below.

These normal forms, like the normal forms of relational data management, often provide useful guidance in the design of model banks, but there are circumstances in which it is reasonable to violate them. For example, corporate financial simulations calculate the values of revenue and expense categories that would result from the implementation of corporate financial or marketing strategies.[16,26] These simulations contain many transitive dependencies, in the form of column totals, row totals and subtotals, financial ratios, and the like. Yet it is not clear that users should think of such a model as a sequence of two models, one of which calculates the detailed data and the other the summary and financial statistics, and corporate models are certainly not programmed in such a fashion. Thus, in structuring a model bank according to one set of criteria, such as the elimination of a certain type of anomaly, one should not lose sight of other important criteria, such as the efficiency of processing or psychological and esthetic criteria.

RELATIONAL COMPLETENESS OF MODEL QUERY LANGUAGES

The need for a convenient user interface is well recognized in the model management literature.[17,18,31,33] It has even been proposed that a dialogue management system, separate from the data and model management systems of a DSS, be implemented to manage all of the interactions between a DSS and its users.[29,30] However, we are concerned here only with the criteria for relational completeness in model management[7] and with the properties of a relationally complete model query language.[4]

The only operation in relational data management that is also useful in model management is selection. The other relational operations of projection and join are not needed. Projection is not needed because the only projections performed with a model are those in which each input attribute is either included in the projection or is specified by a previous selection operation, since a DSS user will not wish to know the output of a model without knowing all of the inputs. This kind of projection can be accomplished by a selection operator and a report writer that prints only those attributes that are of interest. The join operation is not needed, because in relational model management joins can be implemented by the model management system, hence they are transparent to the user. This is explained in the following section. The set and maintenance operations are not needed, because model relations are not union compatible (i.e., a model bank will not contain two models with identical input and output attributes).

In addition to selection (called "execution" in relational model management), there are two further criteria for relational completeness. The first is optimization: the user identifies a nonvoid subset of the input attributes of a relation, a single output attribute, and a maximum or minimum designator. The result of this operation is a relation whose tuples contain the optimal values of (1) the identified input attributes and (2) the output attributes for all values of the remaining input attributes. The second criterion is sensitivity analysis. This is not a relational operation—that is, the output is not a relation but rather a set of sensitivity measures (e.g., partial derivatives or increments in one attribute resulting from increments in another attribute) of an output attribute with respect to an input attribute. More complex sensitivity measures exist and have been described.[4,7,8]

The articulation of these criteria has led to the design, but not yet the implementation of MQL, a SEQUEL-like relationally complete model query language.[4] Consider for example, the relation $REL = \langle P, R; E, N \rangle$, whose input attributes are a sale price P and a raw-material price R, and whose output attributes are the resulting raw-material expense E and net income N. To calculate the sale price that will maximize net income when the raw material price is 5, we write

$\underline{\text{MAXIMIZE}}\ N$
$\underline{\text{OVER}}\ P$
$\underline{\text{WITH}}\ (R, =, 5)$
$\underline{\text{USING}}\ REL$
$\underline{\text{PUT}}\ RPT$

where RPT is the name of a report writer. (The reserved words are underlined.) To find the optimal value of P for each value of R, we eliminate the $\underline{\text{WITH}}$ statement.

A more interesting operation is an embedded optimization. Suppose a raw-material supplier with access to this model (i.e., a model of his customer) wishes to find the value of R

that would maximize E (the raw-material expense of the customer, hence the revenues of the supplier), assuming that the customer will then set its sale price to maximize its net income for that raw-material price. The request statement is

```
MAXIMIZE E
OVER R
USING
  MAXIMIZE N
  OVER P
  USING REL
PUT RPT
```

This type of operation, which is a relational mapping, is of special interest because of its self-referential property. In this respect it is similar to an unofficial test query often used in relational data management: "Find all employees who earn more than their supervisor," applied to a data relation whose tuples contain an employee identifier, the employee's salary, and the identifier of the employee's supervisor. Embedded optimization of this type may perform a similar function in relational data management, namely to serve as an unofficial test query to determine the ease of use of a model query language.

Three linguistic properties of MQL have been identified: it is context-free, it is of star height one, and, although it is of infinite order, a simple extension of the language (to allow for an unlimited concatenation of request statements in a single sentence) is of first order and so possesses the finite-power property. These properties suggest that MQL will not be a difficult language to implement and use.

IMPLEMENTATION ISSUES

Implementation issues have not been adequately addressed in the literature. Questions of security, integrity, query optimization, and the like have received at best passing reference. One issue that has received brief treatment is the implementation of joins.

In the previous section, we used *REL* to denote a relation accessed by MQL, with the implicit assumption that *REL* denoted a model. However, it may denote a set of models that are collectively needed to respond to a query. There are three ways in which the set of models may be accessed. First, if the models are independent—that is, if there is no input to any of the models that is also an output of another model—then the models can be executed in any order and the values of their output attributes assembled to produce a report. Second, if there is at least one input to a model that is an output of another model, but the models, ordered by their input/output relationships, form a partially ordered set (i.e., there are no cycles), then there is at least one sequence in which the models can be executed. In both of these cases implementation is straightforward: the models are executed in an appropriate sequence, with data files presumably used as a medium of communication.[11,20]

The difficulty occurs in the third case, in which the set of models needed to respond to the query is not partially ordered. Consider, for example, two economic models, a supply model and a demand model. The input to the supply model is a vector of supply quantities (presumably for several products in several regions), and the output is a vector of prices at which suppliers will find it economical to supply these quantities; the input to the demand model is a vector of prices to be charged for the products in the regions, and the output is the vector of consumer demands that will be realized at these prices. In general there will be other inputs to the models (e.g., levels of economic activity that affect the supply and/or the demand for the products) and possibly other outputs as well. A consistent set of product prices and product quantities is obtained by entering a proposed value for one of the two vectors (e.g., the prices) into the appropriate (demand) model, calculating the output (demand vector) and entering this into the other (supply) model to calculate its output (vector of supply prices). The process is continued until the vector of supply prices calculated at the end of the process is sufficiently close to the set of demand prices used to initiate the process.[21]

There are four issues to be dealt with in regard to the implementation of joins in this case:

1. *Identification of cycles.* The first issue is the determination of whether there are any cycles in the set of models and, if so, what model attributes (e.g., either the prices or the quantities in the previous example) must be removed and used in the search for a consistent solution. If there are only a few models in the set, this can be determined by inspection, and methods exist for making the determination in more complex cases.[7]
2. *Algorithm development.* The second issue concerns the development of an algorithm for adjusting the appropriate attribute values at each stage of the iterative process (in the previous example, the rule for adjusting the prices to be entered into the demand model). In practice, this appears not to be difficult. A supply/demand model configuration of the type described above, with substantial nonlinearities, has been found to converge in six to ten iterations using a simple adjustment rule.[21] However, there are a variety of sophisticated fixed-point algorithms that may be useful whenever the simple rules are ineffective.[20,27]
3. *Existence.* The third issue is the existence of a consistent solution for the set of models (i.e., the existence of fixed points). A generalization of the Brouwer fixed-point theorem states that a continuous mapping from a convex-compact (i.e., closed and bounded) subset of R^n into itself contains at least one fixed point.[28] Unfortunately, these conditions are often not met in model management. For example, production cost may be a discontinuous function of production quantity, a Boolean variable may represent the decision to invest or not to invest, and many variables—such as costs and revenues—are not bounded. Little work has been done to establish existence conditions for more general cases such as these.
4. *Uniqueness.* The final issue is the uniqueness of consistent solutions for the set of models (i.e., the uniqueness of fixed points). This is important, because lack of

uniqueness means that a set of deterministic models may be a nondeterministic set, and the users of a DSS should be made aware that there may be more than one possible outcome of their decisions.[5] A classical uniqueness result, known as the contraction mapping theorem,[28] does not appear to be useful here. More recent results, based on monotonicity conditions[2] and curvature conditions (in which certain variables exhibit consistent increasing or decreasing returns to scale)[5] are relevant, but the work in this area is far from complete.

One can best summarize the state of the art in the implementation of relational model management systems by saying that much needs to be done, not only in the practice of implementing these systems, but also in the development of theories that will guide implementation.

INFORMATION MANAGEMENT IN DSS

It was suggested in the Introduction that one reason for viewing a model bank as a set of relations is that such a view may eventually be combined with the established relational view of data to yield a unified framework for information management in DSSs, in which the information of interest may be retrieved from a file or calculated by a model solution procedure.[8] With regard to the user view of information (as opposed to the obvious differences in system implementation), data and models compare as follows:

1. *Organization*.—The normal forms for data management and model management, although not identical, are quite similar. This is surprising, because the criteria for constructing normal forms for data relations—the elimination of update anomalies—do not apply in model management. Even so, the anomalies encountered in updating tuples in a data relation do not differ substantially from those encountered in processing a model relation.
2. *Relational completeness.*—As might be expected, the criteria for relational completeness in model management are not at all like those for data management. The only operation they have in common is selection. It is interesting to note that relational mapping appears in both data and model management, but for different reasons. In data management it is used to execute joins, in model management to execute embedded optimizations.

One may ask whether a relational framework is the most appropriate one in which to achieve a synthesis of data and model management—especially since the CODASYL framework also has been extended to support model description and manipulation.[24,32] The advantage of a relational framework in model management and, more generally, in information management is the same as the advantage in data management: its simplicity and elegance. The mathematical principles that lie at the heart of relational algebra are the results of centuries of attempts by natural philosophers and scientists to understand the world around them, and it would not be surprising to find that such a framework is also useful to managers and staff analysts who wish to understand and control their world. A final determination of the usefulness of this framework must await the implementation and operation of model management systems and their integration with data management systems, but this appears to be a good starting point for the integration of information management in DSSs.

REFERENCES

1. Alter, Steven L. *Decision Support Systems: Current Practice and Continuing Challenges.* Reading, Mass.: Addison-Wesley, 1980.
2. Ahn, Byong-hun and William W. Hogan. "On Convergence of the PIES Algorithm for Computing Equilibria." *Operations Research,* 30 (1982), pp. 281–300.
3. Blanning, Robert W. "A Decision Support Language for Corporate Planning." *Policy Analysis and Information Systems,* to appear (1983).
4. Blanning, Robert W. "Language Design for Relational Model Management." In S. K. Chang (ed.), *Management and Office Information Systems.* New York: Plenum Press, 1982.
5. Blanning, Robert W. "The Existence and Uniqueness of Joins in Relational Model Banks," Owen Graduate School of Management, Vanderbilt University, Nashville, 1982.
6. Blanning, Robert W. "Normal Forms for Relational Model Banks," Owen Graduate School of Management, Vanderbilt University, Nashville, 1982.
7. Blanning, Robert W. "A Relational Framework for Model Management in Decision Support Systems." *DSS-82 Transactions,* June 1982, pp. 16–28.
8. Blanning, Robert W. "Data Management and Model Management: A Relational Synthesis." *Proceedings of the 1982 Southeast ACM Regional Conference,* April 1982, pp. 139–147.
9. Blanning, Robert W. "Ambiguity and Paraphrase in a Transformational Grammar for Decision Support Systems." *Proceedings of the Fifteenth Hawaii International Conference on System Sciences* (Vol. 1), 1982, pp. 765–774.
10. Blanning, Robert W. "Model Structure and User Interface in Decision Support Systems." *DSS-81 Transactions,* June 1981, pp. 1–7.
11. Blanning, Robert W. "Model-Based and Data-Based Planning Systems." *Omega,* 9 (1981), pp. 163–168.
12. Blanning, Robert W. "The Functions of a Decision Support System." *Information and Management,* 2 (1979), pp. 87–93.
13. Boer, Germain. "A Beginner's Guide to EMPIRE." Applied Data Research, Inc., Princeton, N.J., 1980.
14. Bonczek, Robert H., Clyde W. Holsapple, and Andrew B. Whinston. *Foundations of Decision Support Systems.* New York: Academic Press, 1981.
15. Bonczek, Robert H., Clyde W. Holsapple, and Andrew B. Whinston. "The Evolving Role of Models in Decision Support Systems." *Decision Sciences,* 11 (1980), pp. 337–356.
16. Boulden, James B. *Computer-assisted Planning Systems.* New York: McGraw-Hill, 1975.
17. Elam, Joyce J. "Model Management Systems: A Framework for Development." *Proceedings 1980 SEAIDS,* February 1980, pp. 35–38.
18. Elam, Joyce J., John C. Henderson, and Louis W. Miller. "Model Management Systems: An Approach to Decision Support in Complex Organizations." *Proceedings of the First International Conference on Information Systems,* December 1980, pp. 98–110.
19. Execucom. "An Introduction to Computer-Assisted Planning Using the Interactive Financial Planning System," Austin, 1978.
20. Filius, Lidia. "Combinatorial Fixed Point Algorithms." In O. Moeschlin and D. Pallaschke (eds.), *Game Theory and Related Topics.* Amsterdam: North-Holland, 1979, pp. 165–172.
21. Hogan, William W. "Energy Policy Models for Project Independence." *Computers and Operations Research* (1975), pp. 251–271.
22. Keen, Peter G. W., and Michael S. Scott Morton. *Decision Support Systems: An Organizational Perspective.* Reading, Mass.: Addison-Wesley, 1978.
23. Keen, P. G., and G. R. Wagner. "DSS: An Executive Mind Support System." *Datamation,* Vol. 25, No. 12 (November 1979), pp. 117–122.
24. Konsynski, Benn B. "On the Structure of a Generalized Model Manage-

ment System." *Proceedings of the Fourteenth Hawaii International Conference on System Sciences* (Vol. 1), January 1981, pp. 630–638.

25. Konsynski, Benn, and Dan Dolk. "Knowledge Abstractions in Model Management." *DSS-82 Transactions,* June 1982, pp. 187–202.
26. Naylor, Thomas H. *Corporate Planning Models.* Reading, Mass.: Addison-Wesley, 1979.
27. Scarf, Herbert E. *The Computation of Economic Equilibria.* New Haven: Yale University Press, 1973.
28. Smart, D. R. *Fixed Point Theorems,* Cambridge: Cambridge University Press, 1974.
29. Sprague, Ralph. "A Framework for the Development of Decision Support Systems." *MIS Quarterly,* 4 (1980), pp. 1–26.
30. Sprague, Ralph H., Jr., and Eric D. Carlson. *Building Effective Decision Support Systems.* Englewood Cliffs, N.J.: Prentice-Hall, 1982.
31. Sprague, Ralph H., Jr., and Hugh D. Watson. "Model Management in MIS." *Proceedings of the 7th National AIDS,* November 1975, pp. 213–215.
32. Stohr, Edward A., and Mohan Tanniru. "A Database for Operations Research Models." *Policy Analysis and Information Systems,* 4 (1980), pp. 105–121.
33. Will, Hartmut J. "Model Management Systems." In E. Grochla and N. Szyperski (eds.), *Information Systems and Organization Structure.* Berlin: Walter de Gruyter, 1975, pp. 467–482.

Focal points for DSS effectiveness

by CARL HARRINGTON
Nationwide Insurance Company
Columbus, Ohio

ABSTRACT

The primary emphasis in this presentation is to take a top-down view of a decision support system (DSS) and to see how the mere existence of DSS causes improvement in management effectiveness. The areas covered include development of information needs, requirement of consistency in using information, discipline in decision making, and the integration of decisions toward goals.

The improved effectiveness from these areas is inherent in the information requirements that are set up during the development of a DSS. Can there be any doubt that improved information consistency and discipline in decision making improves management effectiveness? In other words, no matter the computational advantages and the efficiency of computers to manipulate and display numbers, a DSS automatically results in the identification of issues, begs for the resolution of those issues, and thus produces more consistent and integrated decisions in the choices of action plans and programs for the achievement of corporate goals.

MANAGEMENT EFFECTIVENESS THROUGH DSS

For my presentation I have selected what I believe to be some of the more critical areas that will improve management effectiveness through decision support systems (DSS). These areas are identified by viewing the decision process and the DSS as a totality for the organization rather than as a single functional decision area. Looking at the decision process in this way brings to light many of the attributes of a DSS that are not present or not nearly as effective before a DSS is implemented. The focal points for DSS effectiveness from this viewpoint are

1. *Development of information needs* for decision making
2. *Consistency in using information* by more managers
3. *Discipline in decision making* through a well-constructed process
4. *Integration of decisions* toward corporate goals

As I review each of these focal points I'll present an example of how a Nationwide DSS application improved management effectiveness, and I'll wind up with a few conclusions.

Development of Information Needs

First, DSS improves management effectiveness by focusing attention on the development of information for making decisions. You could say with a great deal of confidence that to manage effectively is to manage with adequate information. Here's where computers, computer software, and the design of a DSS provide the basis for improved management effectiveness.

Useful information, when it can be made available at an acceptable cost, will be used. In strategic or long-range planning before the availability of a computer support system, much of the desired information could not realistically be obtained. Nor was it practical to develop many alternative action plans as the basis for evaluating the potential-effects bottom line.

With the what if and goal-seeking capability of a DSS, alternatives can be discussed, information can be obtained, and choices can be made in less time than it would take to do the traditional one-plan-only planning.

Can there be any doubt that effectiveness is enhanced when management can get definitive answers to their what-if questions before making decisions on what goals should be established? Our experience is that the answers change opinions, and final decisions change a high percentage of the time.

In one of our operating units, when the usual projections had been done manually and goals established by intuition and judgment, the five-year plan called for a compound growth of 11%. When the computer support system was established, the possible growth trend could be developed in different ways more quickly than one projected case could be developed before. Now the manager could look at

1. compound growth
2. linearly accelerated compound growth
3. linear interpolation

and determine the most realistic goal based on the most realistic growth trends. With the additional information the growth rate was adjusted by year to a new goal that was higher than it was before. This improved management effectiveness by providing valuable information previously not available.

Consistency in Using Information

Now let's look at the advantages a DSS brings to management decision making in terms of consistency in use of information.

One of the most frustrating problems of management in reviewing results, in reviewing proposed projects, and in evaluating performance is to have two different figures from two sources for the same item. Much of this is due to different definitions of certain terms and different databases developed to answer the same question or to provide the same information.

A DSS by the very nature of its construction must

1. have one definition for a specific term
2. have the same database from which to get information
3. follow the same sequence of using input
4. produce output in a standard form

All of these DSS characteristics cause management to look at definitions, use of data, input requirements, and how output is best provided to be useful to management.

In many instances, initial problems are created because different management groups have historically used different definitions and many times have developed their own databases to provide their own functional reports.

At Nationwide, when we started developing the strategic planning model, we developed a set of terms and equations to be used for strategic planning. The overall design used a diagram and set of insurance equations for which each functional office provided input. The result was that we resolved several differences of opinion on definitions that had lingered for years. For instance, the equation for earned premium for strategic planning was uniformly used company-wide for the first time.

Discipline in Decision Making

Next let's consider discipline. This factor in many ways is similar to the effectiveness improvement caused by forcing greater consistency. Discipline has an additional dimension, however, in that it is self-imposed on the decision maker, whereas consistency is caused by the influence of others or by a group as a whole.

Managers vary a great deal in the discipline they bring to most aspects of their jobs. But, decision making may be the least disciplined if there is no special structure or framework established to provide the discipline of orderly consideration of all important facts. Again, it is hard to refute the improvement in management effectiveness when orderliness and a structure for reviewing all important information are brought into the picture, and that is what DSS does.

For example, our Marketing Office for years annually distributed what was called the marketing direction. This document provided the rest of management with the outlook for the next three to five years. It became the basis for the planning and administrative decisions of other offices. The marketing direction included such items as premium goals by product, mix of business, sales manpower, turnover in the agency, and training requirements for new agents.

Before a DSS was developed the markets research and product managers extrapolated and adjusted one plan to come up with the marketing direction based on the pertinent facts they had. When the DSS for developing the marketing direction was initiated as part of strategic planning, each variable was identified and historic information developed. It turned out that the information input to produce the marketing direction was better organized and the basic decisions were focused on arriving at agreement on the parameters that were manageable. These included any loss of policies in force, the new sales of agents with varying lengths of service, and desired growth rates relative to the market.

Some significant improvements were made in the decision process used to arrive at the marketing direction.

1. Information requirements were developed on all elements of the equation and projected for reasonableness *before* the decisions were made on premium goals.
2. The process for developing and getting agreement on the direction was more logical and systematic.
3. A more defensible explanation of the goals was established because each of the elements was covered as interrelated parts of the whole. Therefore, understanding and acceptance by other functions was improved.

This caused a discipline in decision making that had not existed in the past. The end-product goals in any given year might not be very different but the understanding and discipline involved in the process of deciding on the goals must definitely enhance the management effectiveness. This would not have happened without the DSS.

Integration of Decisions

Last, let's cover integration of decision making. I guess better integration of decision making is a natural outgrowth of better information, more consistency in use of information, and better discipline in the decision process. But why does DSS cause better integration?

The main reason is that the whole system is designed to achieve a specific objective. The attention to detail and the focus of the DSS on providing information for a specific decision or series of decisions causes an integration of the parts into a meaningful whole. This is not to say manual or other systems don't do this, it is just to point out that DSS improves on the integration of information for decision making.

There are innumerable examples of how a DSS eliminates small errors. Each error on its own is a small thing, but this gets back to consistency of definitions. Once a DSS is in place for the business' information hierarchy there are no longer inconsistencies in aggregations, and to get consistent aggregations there must be better integration of decisions. The parts of an organization are no longer independent parts; they are a subset of the total DSS, which includes overall goals, plans, definitions, parameters, equations, and so on. The end result is a better integration of the various parts of the organization in making those decisions that contribute to achievement of the corporate goals.

Illustrations

A little more can be shown, in the way of example and illustrations, of the enhancements that a DSS can make toward the effectiveness of management.

I have mentioned our strategic planning activities mostly in the previous examples. These improvements come primariliy in an area where there was not much activity in the past. This in many ways made improvements easier to introduce because nothing was being replaced.

In our operational planning, however, this was not the case. Budgeting and annual planning had been around for years and in many ways were a security blanket that managers did not want to give up.

The point of mentioning this difference is that the same forces for improvement are evident in the latter situation. When an old tool or friendly (mostly manual) system is replaced, the potential exists for improvement in management effectiveness. When the DSS becomes available the improvements are so obvious that the enhancements to management effectiveness are forced or readily accepted by the demand for the system.

We consider that we have now developed both a supply push (the strategic planning DSS) and a demand pull (the operational planning DSS). There are a few smaller projects that have been completed where the demand pull comes into play.

1. We developed a reporting system for our commercial underwriting information in about 30 days;
2. We developed a marketing growth model to test market options in about two days;

3. We are constantly adding modules to our financial evaluation model on demand (many times with a few days notice).

Each of the DSS applications have the elements for management effectiveness enhancement through DSS.

CONCLUSION

The value of DSS to enhancement of management effectiveness comes about through the dynamics of the total organization. When decisions are viewed from the perspective of the total organization there are many ways that individual management decisions are forced or motivated toward better development of information needs, consistency in using the information, discipline in following a well-structured decision process, and integration of decisions toward better individual contributions to the achievement of the corporate goals.

DSS can take a lot of credit for enhancement of management effectiveness, not only because the system is so much more effective, but also because it allows the natural motivation and dynamics of decision making to be more obvious, understandable, and acceptable.

Information resource management for corporate decision support

by WILLIAM H. GRUBER
Research and Planning, Inc.
Cambridge, Massachusetts
and
GEORGE SONNEMANN
Nationwide Insurance Company
Columbus, Ohio

ABSTRACT

Decision support systems were a major force that ended the monopoly of information processing resources once held by the traditional corporate data processing function. In order to gain direct access to the data and models that the decision makers needed but could not get from data processing, users increasingly turned to the microprocessor and the use of timesharing from outside vendors. This created a proliferation of information technology to support management decisions. During the very early history of this move toward decision support systems, the corporate data processing function remained focused on its initial mission of processing accounting transactions.

This paper traces the next stage in the management of decision support systems, whereby both corporate data processing, or the information systems function, and the user share the responsibility for decision support as a team. This new integration of the information systems function and users provides an extraordinary improvement in the effectiveness of decision support via the sharing of data and modeling; the integration of decision makers' and corporate functions; and the teaming of information systems professionals knowledgeable in the technology with users who are experts in the management of the business. A case study demonstrates the effectiveness of this integration between information systems and the users and the management of corporate decision support services.

PROLIFERATION OF INFORMATION RESOURCES

During the last decade, there has been a proliferation of information technology to support management decisions. The field of data processing has evolved from one in which the primary focus was the processing of accounting transactions to one that focuses on support for management decisions and other kinds of professional and secretarial functions. This proliferation of information resources has resulted in a number of new concepts and acronyms such as decision support systems (DSSs), management support systems (MSSs), executive information systems (EISs), the information center, end-users, hybrids, executive information room, office technology and office automation (OT/OA), teleconferencing, graphics, micros, and the micro mainframe.

Thus the power of the technology to support management decision making has improved at an extraordinary rate during the last decade. The proliferation of information resources to support management activity has been purchased at a huge expense by many companies. It has been largely uncontrolled in the vast majority of the companies with which we have worked. The management issues created by the proliferation of DSS activity are listed in Table I.

TABLE I—Management issues in corporate decision support systems (DSSs)

1.	Management responsibility for the corporate decision support systems capability
2.	Adequacy of DSS by major corporate function/decision category
3.	Data adequacy/access/security
4.	Sharing of DSS resources
5.	Hardware/software resources
6.	Staff resources
7.	Security for DSS models

INFORMATION RESOURCE MANAGEMENT

It is now timely to focus on the role of information resource management (IRM) for corporate decision support. Out of the chaos that resulted from the proliferation of information technologies has emerged this concept of information resource management. The primary mission of information resource management is the effective use of information resources. Information is today one of the largest categories of corporate assets. In fact, it can be said that information and people are the two most important kinds of corporate assets. In response to the extraordinary importance of information and the very large investments in information assets and information processing, leading companies have implemented information resource management to achieve greater productivity and more effective use of information resources. The scope of the IRM function can be seen from the list of its responsibilities in Table II.

TABLE II—Information systems management responsibilities

1.	Hardware
2.	Software
3.	Information-based company products
4.	Information-based customer relations
5.	Traditional application systems
6.	Technical staff
7.	Data and information management
8.	Telecommunications
9.	Office technology/office automation
10.	Corporate policies/procedures
11.	Micros
12.	Graphics
13.	Executive information room
14.	Information center
15.	Decision support systems
16.	Management support systems

This transformation from traditional data processing to information resource management requires new roles for professionals who have been trained in data processing technology.[1] The information management function of the 1980's, in contrast to the ADP/EDP function of the 1950's and 1960's or the information systems function of the 1970's, proactively assists business managers in improving their effectiveness through improved use of information resource management—the decision support and management support roles of the modern corporate information management function.

USING A DSS

The initial history of the use of DSS is one that Peter Keen has called "cherry picking." Relatively simple problems were solved using inexpensive technology such as Apples and Visi-Calc. The more difficult management problems involving strong data architecture, sizeable databases, links to the corporate management information systems, and integration among several corporate functions were not attempted. In Figure 1 we present a more complete structure of a decision support system than was used in most of the early applications of DSS technology. In this view of DSS, the manager has a

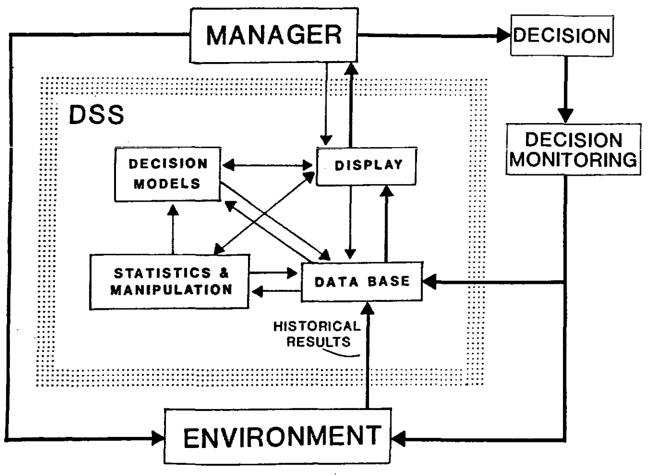


Figure 1—Using a DSS

direct link to a CRT terminal that provides links to databases, linkage to the external environment of the company, and high-level software supporting statistics and decision models. A decision made by the manager is sent to a monitoring system that in turn feeds back into the corporate database and also links to consequences in the external environment.

The technology is now available to support the kind of DSS utilization that is illustrated in Figure 1. We find in the better managed companies that there are now a number of powerful decision support systems that are making a fundamental change in the practice of management. These second-generation DSSs were implemented by teams of corporate information managers and the business managers.

CASE EXAMPLE

The experience of Nationwide Insurance Company in the implementation of decision support systems provides a useful case example for this evolutionary movement toward more effective management practices. Nationwide is similar to other companies in the financial services industry in that corporate management must respond to the highly volatile external environment of deregulation and the entry of new competitors from brokerage and banking. Strategic planning has become a much more important management responsibility in the insurance industry, and it is now absolutely essential for insurance companies to have a strong information system to link the external environment to corporate strategic planning; this is then integrated with operational planning (budgeting) and the management information comparing actual experience with budget.

This cycle of the annual planning cycle implemented at Nationwide Insurance Company is diagrammed in Figure 2. In fact, the process of planning continues all year long, and work on the external environment is a management responsibility that continues throughout the calendar year. However, at the start of the annual planning cycle in January, the results of analyses of the external environment become

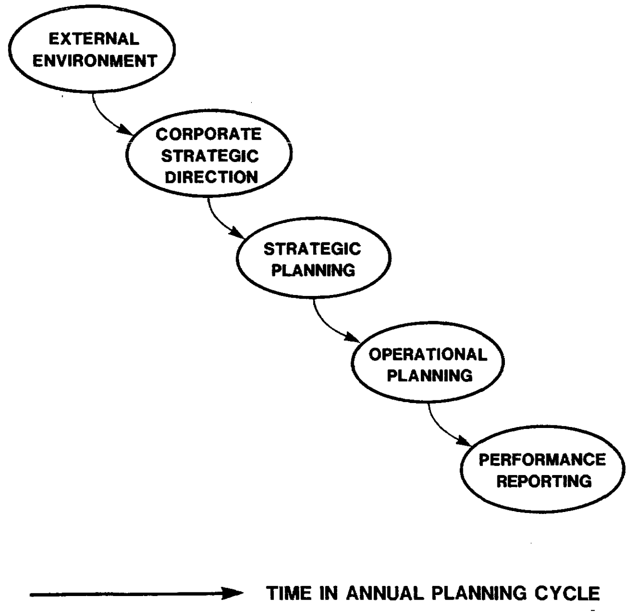


Figure 2—DSS/MIS in the annual planning cycle

inputs into strategic planning; the outputs of strategic planning are key inputs for the operational planning that begins in June. This cycle continues on to the monitoring of the actual performance in the management information module, which, like the evaluation of the external environment, is a continuing process throughout the calendar year.

In order to control the proliferating use of decision support systems, Nationwide initiated a high-level modeling committee with responsibility for the overall fostering and management of decision support systems. This committee was given the mission to authorize investments in the resources needed for decision support systems, such as hardware, software, and staff. The many issues involved, such as data collection, database security, the documentation and security of models, functional integration, and sharing of DSS resources were all

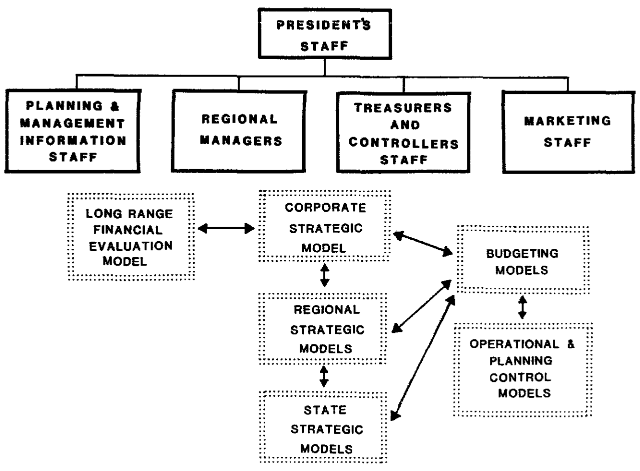

PRESIDENT'S STAFF
PLANNING & MANAGEMENT INFORMATION STAFF
REGIONAL MANAGERS
TREASURERS AND CONTROLLERS STAFF
MARKETING STAFF
LONG RANGE FINANCIAL EVALUATION MODEL
CORPORATE STRATEGIC MODEL
BUDGETING MODELS
REGIONAL STRATEGIC MODELS
OPERATIONAL & PLANNING CONTROL MODELS
STATE STRATEGIC MODELS

Figure 3—Long-range and strategic financial evaluation DSS

part of the mission of the Nationwide modeling committee. The contribution of this Nationwide committee can be seen from the structure of modeling used at Nationwide, which is illustrated in Figure 3. It can be seen that a significant number of Nationwide management functions are now supported by corporate modeling capabilities. These models are processed on hardware dedicated to modeling; the processing uses very effective software packages. There has been extensive education of professional staffs to support modeling, and a sizeable number of business managers have also been trained in the utilization of corporate models.

In the competitive jungle of the financial services industry in the 1980's, this kind of more effective management capability is clearly needed, and the Nationwide experience in rapidly moving forward with this kind of capability indicates that the technology is now available to provide needed management support.

CONCLUSION

We are now moving into an information era in which managers are supported in ways that were rarely attempted even a few years ago. The discontinuity in the external environment of business, the difficult economy, and the rapid progress of information technologies for management and decision support are all factors that have encouraged corporations to invest in this kind of support capability.

The major change that has occurred in the utilization of decision support and management support capabilities during the last five years has been the acceptance in the better managed companies of a new role for information managers. In contrast to even five years ago, when business managers disregarded the skills of data processing experts and did their own thing in decision support, today in the better managed companies there is an impressive integration of information management and business management to achieve the more effective use of information resources in support of management decisions. The experience achieved in leading companies has demonstrated the feasibility and effectiveness of this integrated strategy for using modern information resources. The experience of the last few years has been sufficiently impressive to justify a forecast of very rapid acceptance of information resource management for decision support and management support during the 1980's as companies improve in their ability to use modern information resources.

REFERENCES

1. Synnott, William R., and William H. Gruber. *Information Recources Management: Opportunities and Strategies for the 1980's.* New York: John Wiley, 1981.

Developing a strategy profile for management support systems

by GARY K. GULDEN and EEVELYN S. ARKUSH
Index Systems, Inc.
Cambridge, Massachusetts

ABSTRACT

In most business organizations today, a new era in the application of information-systems technology has been entered: The era of management support systems (MSS). In fact, there is much evidence to support the claim that management support systems (as distinct from conventional transaction processing or MIS systems) is the fastest growing segment of the information systems portfolio. This paper explores the urgent need for MSS planning and presents a practical methodology to approaching this very different planning problem.

THE EXPLOSIVE DEMAND FOR MANAGEMENT SUPPORT SYSTEMS

In most business organizations today, a new era in the application of information-systems technology has been entered: The era of management support systems (MSS). In fact, there is much evidence to support the claim that management support systems (as distinct from conventional transaction processing or MIS Systems) is the fastest growing segment of the information systems portfolio.

The pressures behind the explosive growth in demand for management support systems are severalfold:

1. First, managing in the eighties presents more challenges to executives than ever before. The pace at which business is conducted and the speed with which the competitive and economic environment can change are extraordinary, even by the standard of 10 or 15 years ago. New and complex businesses hit the ground running at a scale unheard of even a decade or two ago.
2. Managers in today's environment recognize that improved access to information regarding their marketplace, and their own performance within it, are essential for success and may even represent sources of strategic gain.
3. With the availability of fourth-generation MSS tools and powerful personal microcomputers, managers' frustrations over the failure of traditional (efficiency-oriented) systems to be able to respond to their effectiveness-oriented requirements are being offered a powerful and constructive outlet.
4. Finally, demand for management support systems has been heightened by the fact that, in many organizations, the vast majority of information systems resources are being cornered in order to replace 10- or 15-year-old transaction-based systems—in many cases, the same systems that promised at their inception, but never delivered, valuable management information!

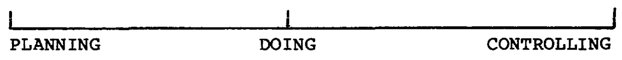
THE PUSHERS AND THE CONTROLLERS

In the face of this explosive demand growth, information-systems managers are exhibiting two very different kinds of behavior: Pushing and controlling.

1. Those who are pushing correctly perceive that the demand for management support systems is strong, valid, and won't go away. Thus, they have taken the posture of aggressively providing and promoting the use of MSS technology.
2. Those who are controlling correctly perceive that the explosive growth and demand in management support systems carries with it the unpleasant opportunity to learn all over again the uncomfortable lessons of the stages-of-growth problems they experienced over the last two decades. Thus, they have taken a posture that applies a go-slow attitude, coupled with a variety of standards and rules for the purchase and use of MSS technology in their organizations.

The contrast is clear: Pushing organizations are characterized by lots of *doing* in the MSS area but not much *managing*; where controlling is the approach, there is lots of *managing*, but not much *doing*! What is most noteworthy, is that in *neither* case is there much overall *planning* going on anywhere!

Both the pushers and the controllers are responding to valid pressures and risks but are failing to recognize that *doing* and *controlling* are separate pieces of an overall process that starts with planning. (See the illustration below.)



PLANNING FOR MSS?

In the world of conventional transaction processing, operational, or management information systems, managers increasingly devote very substantial effort to the development of comprehensive long- and short-range systems plans. But practically no one makes this same sort of effort for MSS planning!

For most organizations, planning for MSS is not a current pressing issue principally because MSS activity and end-user computing typically represent a small proportion of present information-systems expenditures and because the very nature of MSS (its end-user orientation) does not produce much contention for I/S staff resources. MSS hence generates no real pressure for the rationing that is an objective of many systems plans.

But the *growth rate* of MSS and end-user computing activity—once it begins in an organization—is extremely rapid, aided and abetted by a growing supply of "friendly" tools available as mainframe packages on inexpensive personal computers or on outside timesharing services. If one looks ahead, it is not difficult to see the day approaching when, in many organizations, management support systems activity and its attendant end-user computing will represent a clear majority of information-systems expenditure and usage. Even today there are a few information systems/resources managers who have already allowed this sort of growth to occur in a poorly planned fashion and, as a consequence, have either had their management clamp a tight lid on their budgets or have seen the MSS/end-user area taken over entirely by another part of the organization.

The point is, it is essential to have strategies and plans for MSS, and the time to get started on them is now, while there is some hope of keeping at least a half-step ahead of the rising water.

THE ELEMENTS OF AN MSS STRATEGY

Because MSS and end-user computing are so radically different from traditional transaction or MIS systems, it is no surprise that the dimensions of MSS strategies and plans look different also. In developing an MSS strategy, it appears that there are five interrelated strategy elements or areas in which one must make some choices of direction. They are as follows:

1. Marketing—Who are the customers and how do I reach them?
2. Products—What do the customers need?
3. Customer support—What is our approach to delivering and servicing our "products?"
4. Delivery technology—What type of technical environment is to be employed?
5. Management/policy—What are the rules that need to be in place to provide appropriate management control?

For each of the strategy elements, there are one or more sets of ranges or spectra of possible strategic positions, as shown below. (Others may also occur to you as you think through your own situation.)

Marketing

An MSS marketing strategy can be either proactive (with an active marketing program to attract important new customers) or reactive (providing support for those who ask). The spectrum would thus look like this:

PROACTIVE
REACTIVE

The hierarchical target markets, or the potential customers for MSS, tend to fall into three types:

1. Operational decisionmakers who may require support for routine decisions, as in scheduling or purchasing activities, for example.
2. Analytical staff who are providing staff support to executives.
3. Executive decisionmakers who prefer or require support tools for their personal use in managing and decisionmaking.

The spectrum is consequently

OPERATIONAL DECISIONMAKERS
ANALYTICAL STAFF
EXECUTIVE DECISIONMAKERS

and one's MSS strategy needs to recognize the market or markets that are presently being served and that will be served in the future. Additional spectra for target markets may include functional areas (i.e., marketing, finance, etc.) and the organizational areas (corporate, division, subsidiary).

Please note that the spectrum device proposed here is intended only to emphasize a range of possibilities but not an implied direction. Moreover, one's strategy may be at a *point* on the spectrum, a series of points, or within a *band* on the spectrum.

Products

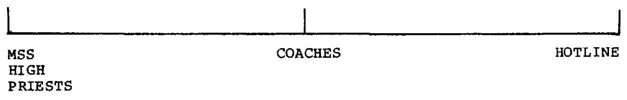
The strategy spectrum for products can be depicted as a range from highly-focused, single-purpose MSS applications at one end, to extensive general-purpose data-plus-software-tools environments at the other.

SINGLE-PURPOSE MSS
GENERALIZED DATA & TOOLS MSS FACILITIES

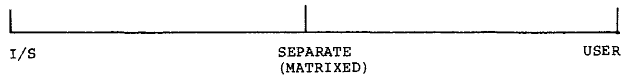
Customer Support

There are two strategy spectra here, one dealing with the nature of the support and the other with its organizational location.

With respect to the nature of support, the spectrum ranges from a cadre of specialists who are proficient in the technology and perform all the applications work themselves (MSS high priests), to personnel who view their roles as teachers and coaches of end users, to people who provide assistance over the telephone (the hotline) and perform no applications work whatever.



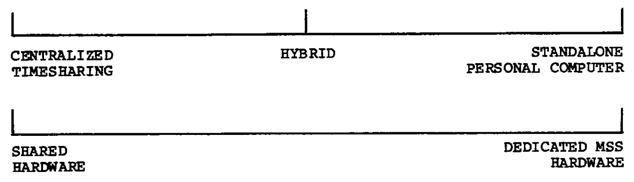
Location of the support may be within the information-systems organization, in a separate group matrixed to I/S and user organizations, or in one or more user organizations.



The information-center concept is a user-support concept that addresses both the type and location of the support and frequently addresses the hardware issue.

Delivery Technology

There are two strategy spectra here concerning the variety and location of the hardware:



Management/Policy Groundrules

There are a variety of possible strategy spectra in this area. The major decision for each depends on the degree of discipline and control that is to be exercised. Management policies and procedures that can be addressed using the spectra include:

1. Data administration
2. Security
3. Development and documentation life cycles
4. Eligibility (application screening) (To separate out applications that may end up as production transaction-based systems.)
5. Cost justification
6. Chargeout

The preceding discussion of strategy spectra may convey, by its brevity, that you should simply sit down and begin to identify the points or bands on each spectrum where you want your strategy to be. Nothing could be farther from the truth. Each decision on strategic positioning should be made after careful fact gathering and thoughtful analysis. What *can* be done fairly quickly, however, is to use the spectra as a tool to record where you stand *today* with respect to each strategy element. That can be a very useful initial diagnostic and descriptive exercise.

An MSS Strategy Profile

Either as part of an initial diagnostic process, or once you have drawn some preliminary conclusions about the individual spectrum point or points at which you wish to direct your MSS strategies, it is helpful to have a way to visualize the collective profile of those individual decisions. Arraying the various strategy elements and their spectra on a single sheet of paper can aid the checking of completeness and internal consistency (see Figure 1). Clearly there is no one right pattern or profile of points or bands, but an unusually skewed or scattered pattern may suggest the value of some additional examination and thought.

Once the strategic destinations, or strategy profile is developed, a plan can be developed to reach this destination. Figures 2 and 3 are two examples of profiles drawn from actual Index client situations, accompanied by some interpretive comments.

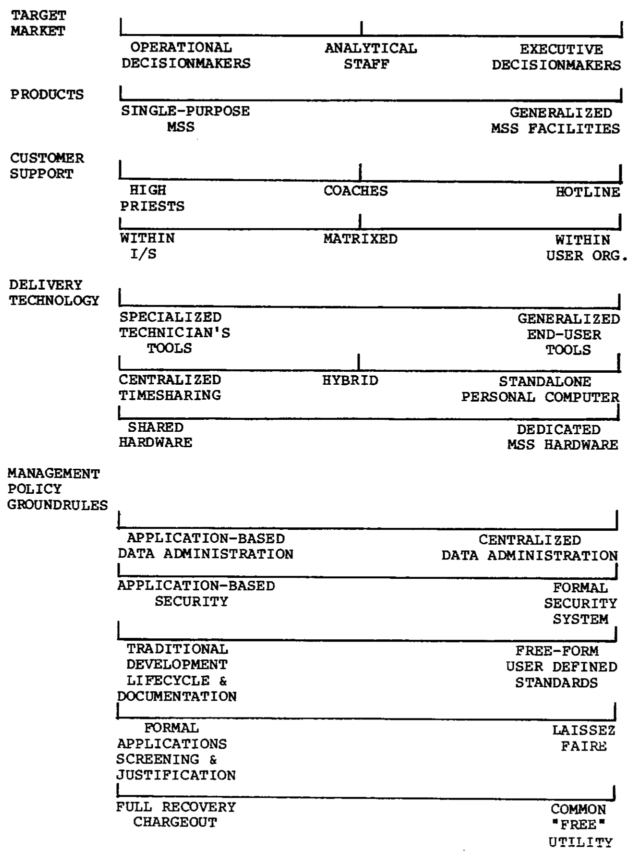

TARGET MARKET
OPERATIONAL DECISIONMAKERS
ANALYTICAL STAFF
EXECUTIVE DECISIONMAKERS
PRODUCTS
SINGLE-PURPOSE MSS
GENERALIZED MSS FACILITIES
CUSTOMER SUPPORT
HIGH PRIESTS
COACHES
HOTLINE
WITHIN I/S
MATRIXED
WITHIN USER ORG.
DELIVERY TECHNOLOGY
SPECIALIZED TECHNICIAN'S TOOLS
GENERALIZED END-USER TOOLS
CENTRALIZED TIMESHARING
HYBRID
STANDALONE PERSONAL COMPUTER
SHARED HARDWARE
DEDICATED MSS HARDWARE
MANAGEMENT POLICY GROUNDRULES
APPLICATION-BASED DATA ADMINISTRATION
CENTRALIZED DATA ADMINISTRATION
APPLICATION-BASED SECURITY
FORMAL SECURITY SYSTEM
TRADITIONAL DEVELOPMENT LIFECYCLE & DOCUMENTATION
FREE-FORM USER DEFINED STANDARDS
FORMAL APPLICATIONS SCREENING & JUSTIFICATION
LAISSEZ FAIRE
FULL RECOVERY CHARGEOUT
COMMON "FREE" UTILITY

Figure 1—MSS strategy-profile worksheet

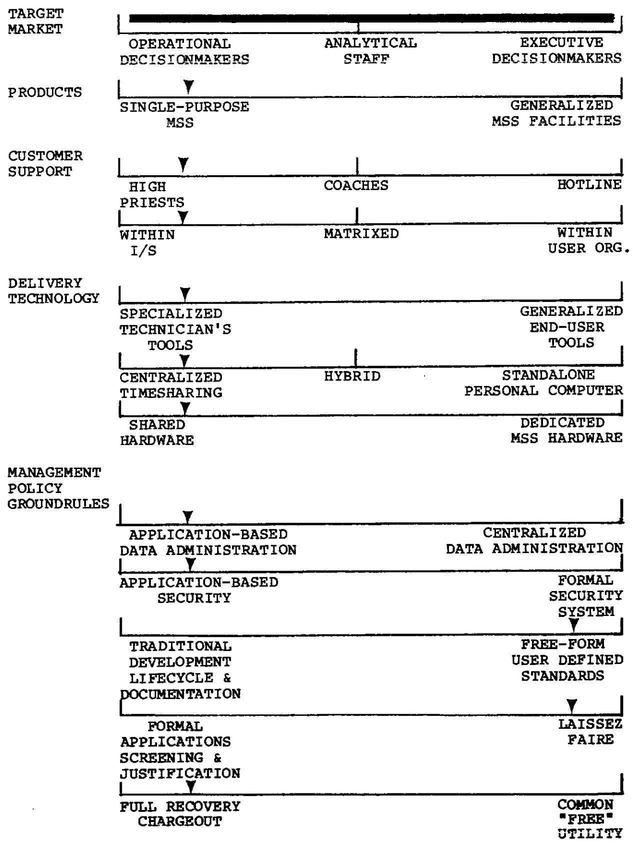

TARGET MARKET
OPERATIONAL DECISIONMAKERS
ANALYTICAL STAFF
EXECUTIVE DECISIONMAKERS
PRODUCTS
SINGLE-PURPOSE MSS
GENERALIZED MSS FACILITIES
CUSTOMER SUPPORT
HIGH PRIESTS
COACHES
HOTLINE
WITHIN I/S
MATRIXED
WITHIN USER ORG.
DELIVERY TECHNOLOGY
SPECIALIZED TECHNICIAN'S TOOLS
GENERALIZED END-USER TOOLS
CENTRALIZED TIMESHARING
HYBRID
STANDALONE PERSONAL COMPUTER
SHARED HARDWARE
DEDICATED MSS HARDWARE
MANAGEMENT POLICY GROUNDRULES
APPLICATION-BASED DATA ADMINISTRATION
CENTRALIZED DATA ADMINISTRATION
APPLICATION-BASED SECURITY
FORMAL SECURITY SYSTEM
TRADITIONAL DEVELOPMENT LIFECYCLE & DOCUMENTATION
FREE-FORM USER DEFINED STANDARDS
FORMAL APPLICATIONS SCREENING & JUSTIFICATION
LAISSEZ FAIRE
FULL RECOVERY CHARGEOUT
COMMON "FREE" UTILITY

Figure 2—MSS strategy profile of company A

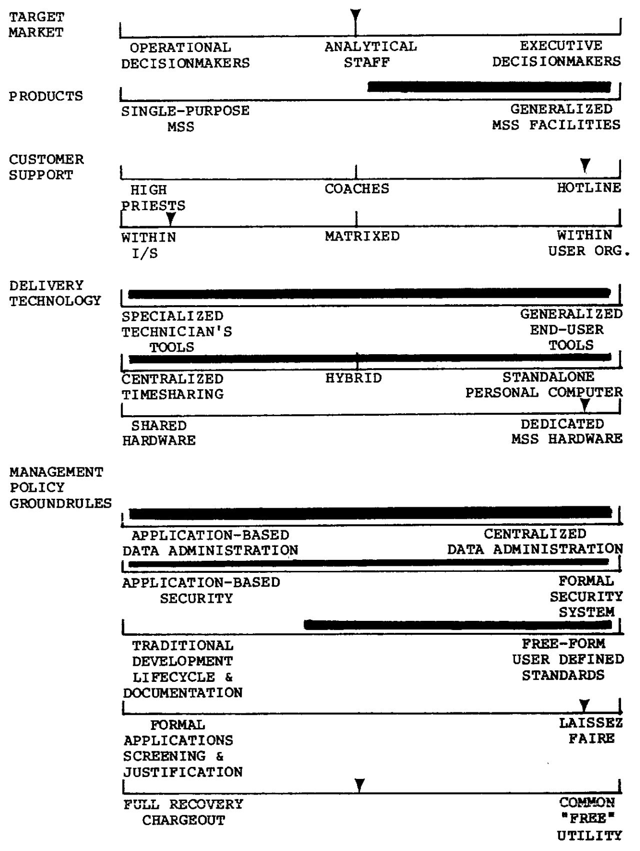

TARGET MARKET
OPERATIONAL DECISIONMAKERS
ANALYTICAL STAFF
EXECUTIVE DECISIONMAKERS
PRODUCTS
SINGLE-PURPOSE MSS
GENERALIZED MSS FACILITIES
CUSTOMER SUPPORT
HIGH PRIESTS
COACHES
HOTLINE
WITHIN I/S
MATRIXED
WITHIN USER ORG.
DELIVERY TECHNOLOGY
SPECIALIZED TECHNICIAN'S TOOLS
GENERALIZED END-USER TOOLS
CENTRALIZED TIMESHARING
HYBRID
STANDALONE PERSONAL COMPUTER
SHARED HARDWARE
DEDICATED MSS HARDWARE
MANAGEMENT POLICY GROUNDRULES
APPLICATION-BASED DATA ADMINISTRATION
CENTRALIZED DATA ADMINISTRATION
APPLICATION-BASED SECURITY
FORMAL SECURITY SYSTEM
TRADITIONAL DEVELOPMENT LIFECYCLE & DOCUMENTATION
FREE-FORM USER DEFINED STANDARDS
FORMAL APPLICATIONS SCREENING & JUSTIFICATION
LAISSEZ FAIRE
FULL RECOVERY CHARGEOUT
COMMON "FREE" UTILITY

Figure 3—MSS strategy profile of company B

Company A

As shown in Figure 2, the approach that Company A is taking is fairly clear. The MSS applications are highly focused and performed by a group of specialists using specialist's tools (a highly-enhanced APL-based tool in this particular case). The company is trying to serve a broad array of customers, however, and that contrasts fairly sharply with its more narrow choices in the other elements. This suggests there may be some potential pressures for change—and in the actual situation there are. Customers at all levels are becoming increasingly frustrated at having to wait their turn and to compete for the attention of one of the MSS specialists. Moreover, the customers are concerned that they are not developing any of their own MSS capabilities and that they lack the sorts of end-user tools that might allow them to do so.

Company B

As shown in Figure 3, Company B has taken the approach of providing a powerful tools-plus-data environment aimed at providing support primarily for staff personnel. The delivery environment offers a broad range of capabilities, and the analysts use it very heavily (volume is growing at about 50% per year and now accounts for about 25%, which is 30 percent of computer usage in the entire company). There are several pressures for change, however:

1. Executive-level personnel are beginning to have difficulty identifying what benefit they have received from their growing MSS timesharing bills. Some are concerned that an expensive "analysts sandbox" has been created, and they are clamping down on computer-usage budgets in response.
2. The people who run the central computer utility are forecasting large, expensive capacity additions but are also being questioned harder about where the benefit is.
3. Operational decisionmakers are feeling poorly supported, because most of the analytical work is in the planning arena.

The pressures for change in this case appear to boil down to the need for diversifying Company B's portfolio of MSS customers and products—particularly to find ways to provide specific high-value support to operating and executive levels.

There are two additional points to make on the use of the strategy-profile technique.

1. The strategy profile worksheet is intended to be flexible and to be tailored to a given organization. Hence the precise definition and number of possible strategy spectra are not fixed and should be altered as appropriate.
2. The strategy profile developed should be used to create a plan for reaching the strategic destinations envisioned. It is also an excellent vehicle to communicate the organization's position on management support systems.

SUMMARY

The central messages of this paper are

1. Management *can* and *should* do planning for MSS and end-user computing.
2. There are at least five major strategy elements for which directional decisions need to be made.
3. It is important to examine these strategy directions in relation to each other to check for internal consistency and to anticipate where pressures for subsequent change are likely to occur.
4. The strategy-profile technique is one way to visualize your MSS strategy and should be used both *diagnostically* (to look at what you are presently doing—your de facto strategy) and *prospectively* (to indicate intended directions or shifts in MSS strategies).

The DSS development system

by ROBERT H. BONCZEK
Purdue University
West Lafayette, Indiana
NASIR GHIASEDDIN
University of Notre Dame
Notre Dame, Indiana
CLYDE W. HOLSAPPLE
University of Illinois
Champaign, Illinois
and
ANDREW B. WHINSTON
Purdue University
West Lafayette, Indiana

ABSTRACT

As decision support systems become more commonplace, the demand for automatic and semiautomatic DSS development systems increases proportionately. Such systems provide a set of tools that guide the construction of models in response to a user's query. This paper describes a set of such tools that provide capabilities for analysis, design, module management, and report and graphics generation.

INTRODUCTION

In recent years, the need for increased productivity in managerial decision-making activities has been felt both in private and in public sectors. This need is mainly motivated by the competitive nature of the business world, which calls for more and more efficiency as an essential ingredient for business survival. Decision support systems (DSS) have been shown to increase management's effectiveness and productivity in handling decision problems. The potential benefits of decision support systems have created an ever-increasing need for these systems. This need has accelerated the efforts to build more and more such systems. As the potential benefits of decision support systems are realized by more decision makers in various fields, the need for such systems will increase even more. In 1978 only 20% of all applications developed were for management control, planning, and analysis (which roughly falls into the area of DSS), while 80% were operational. However, since then this breakdown has changed dramatically and it is estimated[1] that by 1983, 55% of the new programs will be written for management control, planning, and analysis, and only 45% of the new applications will be operational. The foregoing discussion suggests that there is a serious need for many new decision support systems to aid decision makers in various fields.

Although the computer industry now has some 35 years of experience, the process of software development is still slow, difficult, costly, and error-prone. The process of DSS development is no exception to this and perhaps it is even more difficult than the development of many other systems. This is due to the following reasons: (1) many problems that the DSS is intended to help solve cannot by nature be prespecified, (2) the problem itself or the user's perception and/or conception of the problem will change over time, (3) often the user does not know his/her true needs, and (4) the DSS often should support various needs of many users. That is, it should support the solving of many problems through various decision-making styles and in many different problem situations.

The need to find more productive ways of software development in general and application software development in particular is discussed in[2] in some detail. We can identify four basic approaches that yield higher productivity in the process of software development. These are: structured design and programming, higher-level languages and special tools, the use of prefabricated pieces in construction of a new system, and automatic program generation. The complete automation of the software development process is yet a few years away, but it is certainly very desirable to move closer and closer to this ultimate goal. It seems reasonable to assume that the ultimate goal of complete automation will not be reached through one revolutionary step; rather it will happen through many evolutionary steps. Before complete automation is possible, many specialized tools must be developed to facilitate the process of software development through a semiautomatic process.

If we accept the hypothesis that many new decision support systems will be needed in the near future, it seems reasonable to focus all of our efforts on building a facility that will enable us to develop such systems with great efficiency, rather than on building individual systems in the traditional ways. This is the direction that we would like to follow.

This paper discusses the design of an environment for the development of decision support systems. We call this environment the decision support system development system or DSSDS. The system we propose will be a semiautomatic system within which a collection of highly specialized tools will be used to manufacture the individual components of a DSS from prefabricated pieces, from scratch, or from a combination of these two techniques. The individual components then could be assembled to create an integrated system. Moreover, the system would be capable of supporting the product (i.e., the developed DSS) throughout its entire life cycle.

DSS DEVELOPMENT PROCESS

Development of a decision support system requires all phases of a systems' life cycle; however, the iterations between various phases of the life cycle happen at a much faster rate. In other words, since the problem space for a DSS is continually changing, modifications and extensions of a DSS should be regarded as a norm rather than as an exception. This certainly imposes a serious constraint on the development process of a DSS. To deal with this problem we propose the following characteristics as essential features of any DSS development system (DSSDS) that is intended for the production of a successful decision support system.

1. The DSSDS should support quick production of decision support systems.
2. The decision support systems should be produced with inherent features of modifiability as well as extensibility.
3. The DSSDS should support rapid modification and production of extensions to the DSS.

The imposition of the first requirement on the DSSDS stems from a more profound reason than just the productivity gains. Since the problem itself and the user's conception and/or perception of the problem are continually changing, the DSS should be produced rather quickly, otherwise it will be obsolete as soon as the development process is finished. The second requirement simply states that the DSS should be built in such a way that it can be expanded (ideally indefinitely). The third requirement states that the rate of implementation of modifications and extensions should be faster than the rate of generation of needs for new modification, otherwise the system will never keep up with needs of its users and becomes obsolete very soon.

It is obvious that these requirements cannot be satisfied through traditional system development processes. There is a serious need for a more powerful facility to help satisfy these requirements. The design of such a facility will be discussed in later sections, but before that we need to talk about another very important issue in the system development process, which is prototyping.

PROTOTYPING

The key to the development of a successful system is the correct understanding of the problem by the developer. The understanding of the problem takes place in the analysis phase of the system's life cycle. At the end of this phase a formal specification of requirements is written by the analyst, which must then be reviewed and approved by the client before the design can begin. The importance of the requirements specification stems from the fact that experience has shown that errors in requirements specification are usually the last to be detected and the most costly to correct.[3,4] The importance of this stage in the system's life cycle has been well understood from the early years in the field of systems analysis and design. To overcome the problems in requirements specification, various methods and tools have been developed to assist the developer in this stage of the development. The system specification tools such as problem statement language (PSL)[5] and requirements specification language (RSL)[6] will help the analyst in checking the consistency and clarity of the specification.

The main problem with these techniques is that they rely heavily on the user to verify the accuracy and completeness of the problem specification by looking at the formal specification of the requirements. It is often difficult for the users to visualize what they see on paper as solving their problems. Besides, many users, especially the DSS users, do not have a clear understanding of their true needs prior to actual use of the system.

A second group of tools, developed with the realization of the potential difficulty of the users in verifying a printed specification of the requirements, provides graphical means to overcome this problem. Among these tools are structured analysis and design technique (SADT)[7] and SAMM.[8] Graphical representations are usually better understood by the user, provide a better picture of the system the way it has been understood by the developer, and enhance productive feedbacks. However, the user never becomes certain whether a system will satisfy his/her true needs until he/she actually starts using it.

The understanding of the true needs of the user by the developer and the user him/herself can be greatly enhanced through the development of a prototype of the proposed system. Using a prototype the user can more accurately examine whether the right problem is being solved and also if he/she has been understood correctly by the developer. That is, the answer to the two vital questions of the success of the system are provided with the most accuracy possible. The user, by exercising the prototype, can provide vital feedbacks to the developer. These feedbacks can be used by the developer to finalize the requirements specification. By developing a prototype the developer also will experience the difficulties and potential problems in the development process.

Thus it is clear that a prototype is a valuable learning vehicle both to the user and to the developer. In practice, however, prototypes are not built very often because of their high cost of development and also because of the additional time required for their development.

These problems of prototyping could be overcome through the use of a set of powerful tools that facilitate a relatively cheap and speedy development of a prototype.*

The DSS development system to be discussed in the next section will, among other things, provide such tools. Note that the emphasis in prototyping should not be on producing a very efficient system; rather, the emphasis should be on rapid production of a prototype that accurately reflects the requirements of the proposed system as perceived by the developer. A word of caution concerning the development of prototypes is in order: It is often necessary to make changes to the prototype in order to observe the user's reactions to modified versions. These changes should be stopped the moment no new knowledge can be learned from modification, or the cost of modification outweighs the benefits gained from it. In any event, the temptation to carry on the development of the prototype in order to turn it into the delivered system should be strongly resisted.

Prototyping in no way conflicts with the use of other systems-analysis tools and techniques. In fact, we propose that tools should be provided to the developer to help capture pertinent information from the user. This information should be stored in an organized way in a database. Automatic checking of the data's consistency and completeness should be performed, and finally, tools should be provided to the designer so that he/she can retrieve the data pertinent to each operation both quickly and in a convenient format. The developer can use this information to build a prototype with an acceptable level of accuracy for examination by the user. The requirements specification is finalized when the user is convinced the proposed system will indeed satisfy his/her needs.

THE DSS DEVELOPMENT SYSTEM

The DSS development system (DSSDS) is an environment for the development of decision support systems. The environment consists of highly specialized tools to be used by the DSS

*A good example of an existing system for rapid prototyping is KnowledgeMan,[9] which is available inexpensively on microcomputers. KnowledgeMan has facilities for data management, ad hoc inquiry, statistical analyses, spreadsheet analysis, customized I/O screen forms, report management, and model building. Here, we are proposing an even more powerful set of tools.

developer throughout the development process to facilitate the development of a successful system. The DSSDS will increase the productivity of the developer and help him/her to produce with a moderate cost a DSS based on the true needs of the user. The philosophy of the DSSDS is based on two very simple, but also very important concepts: the use of highly automated tools throughout the development process and the use of prefabricated pieces in the manufacturing of a whole piece whenever it is possible. The first concept increases the productivity of the developer in the same way an electric saw improves the productivity of a carpenter using a hand saw. The second concept increases the productivity of the developer analogous to the way a prefabricated wall increases the productivity of the carpenter building a house.

Although many of the tools that DSSDS provides could be used in the development of any application system, our emphasis would be towards tools that are helpful in the development of a DSS in particular. By specializing we expect to gain efficiency in the development process because there will be a real need in the future for the development of many decision support systems. Design of any large system for the first time is a major task. To insure the success of the system, it is not recommended that a gigantic system be designed from the beginning, in the expectation of supporting the development of every detail of the system. Rather, in the design of a DSSDS we follow the evolving characteristic. That is, we think that a nucleus DSSDS should first be designed and developed to support the essential needs of the developer. However, the system should be extensible so other features can be added to it when the need for them becomes apparent. Nevertheless, the DSSDS should have the following characteristics:

1. The DSSDS should support the development of a successful DSS.
2. The DSSDS should support the development process throughout the entire life cycle of the system, that is, it should support capturing of the requirements from the user, development of the prototypes, design and implementation of the delivered system, testing, and finally the maintenance of the DSS.
3. The DSSDS should support the development of different decision support systems in different programming languages and possibly for different target computers.
4. Various tools of the DSSDS should be relatively easy to use and independently available.
5. The DSSDS should be capable of evolving over time.

In this context the independence of various tools implies only the functional independency; however, coordination of various tools is essential. The evolving feature of DSSDS here means that the system should allow new tools to be added to the system as well as allow old tools to be improved or replaced by more advanced tools.

AN OVERVIEW OF THE DSSDS ENVIRONMENT

The DSSDS environment can be thought of as a workshop with many tools and prefabricated parts that the developer can use throughout the process of building a new DSS or to upgrade or repair an existing DSS. The environment of the DSSDS is shown in Figure 1. The developer is provided with a development language (DL), which is basically a powerful command language. Modules can be written in command language or in any other programming languages such as FORTRAN, COBOL, or PASCAL. By module we mean any set of executable lines of code that has a name and is written to do a certain job. A module can be used independently, or it can be used in conjunction with other modules to build a more complex module. A module can do computation, perform read and write operations, transform data, or perform any other computer operations in order to achieve a certain objective.

In addition to the development language, a number of other facilities are available to the developer. These are systems analysis and design facility (SADF), a model management language (MML), a screen management language (SML), a source code manager (SCM), a report generator (RG), a graphics generator (GG), and a request handler (RH). Each of these facilities can be used through the command language or independently.

The Development Language (DL)

The development language is a command language. Its function is to provide a host to other facilities, as well as to provide a collection of useful functions to be used by the developer. Individual commands or procedures written in MML, RG, GG, and so on can be invoked from the DL. The command language provides interface between modules written in various facility languages (i.e., MML, RG, GG, etc.) as well as modules written in programming languages like FORTRAN, COBOL, PASCAL, and so on. In this way the developer can write a program whose components are written in different programming languages and/or use various development facilities. For example, to create a plot of the predicted sale for years YR1 to YR2, the following program can be written:

```
RETRIEVE (SALE, YR, R10)
CALL REGRESS (SALE, YR, COEF)
CR FYR (10) = YR1 TO YR2
CALL FORCAST (COEF, FSALE, FYR)
GG. PLOT (FSALE, FYR)
```

The first line is intended to retrieve the sale values with the corresponding year values (YR), for ten most recent years (R10). The second line will run a regression on sale as a dependent variable against YR. Then, variable FYR is defined to be an array with values YR1 to YR2. The fourth line runs a forecasting model using the coefficients produced in line two. Finally, line five invokes the command PLOT from a graphics generator (GG) to plot the predicted sale against the future years.

By being able to create a program whose components are written in different languages, we benefit in two ways: First, each component can exist in its most efficient form. That is,

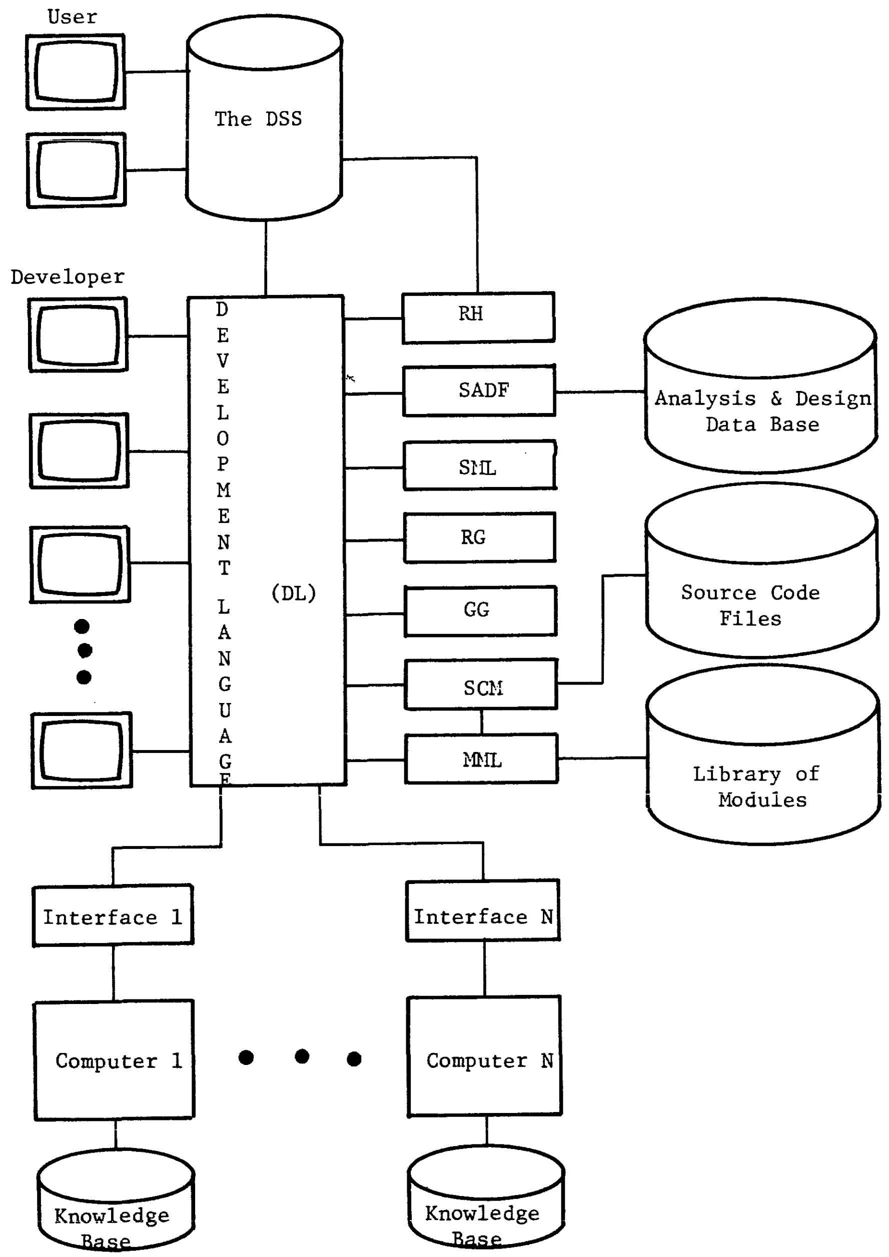


Figure 1—An overview of the DSS development system (DSSDS)

each module can be written in a language that is most suitable to its function. Second, more productivity can be gained by using many existing modules currently available in different programming languages. The purpose of this paper is not to discuss the syntax or semantics of the command language; rather it is to present the concept of such a language. A sample of some other commands is shown in Figure 2.

Systems Analysis and Design Facility (SADF)

Development of a DSS, like any other system, starts with analysis. The aim of the systems analysis phase is to gather enough information about the needs and operations of the system so that any qualified data-processing professional will by reviewing this information be able to understand what the needs and requirements of the new system are and what it is supposed to do. The purpose of the systems analysis and design facility (SADF) is to help solicit pertinent information from the user, to store and organize this information in a database, and to check the consistency of the information and make it available to the developer in a usable form.

Development of any nontrivial information system generally requires the participation of many people. One problem facing the development of such systems is the documenting of

CREATE x	Create a file and call it x; if x is not present a working file is created.
STORE x	Store file x. The system will prompt for the location of the file and security feature. The default for x is the current working file.
SAVE	The system will save the entire work of the session as it is, so it can be continued at a later time.
RECREATE	The system will recreate the working environment as it was left off in the last session.
EXECUTE x, (C = Comp, I = data, O = y)	The system will send module x to computer comp to be executed using file "data" as input file, and sending the output to file y. Defaults are the main frame computer and terminal input/output respectively.
EXTRACT x, y, type	Extract file x and place it on file y. Type can assume values M or D for module and data. If system is unable to find the location of x, prompts for help.
RETRIEVE (x,y,z,...,pi)	Retrieve i instances of variables x, y, z, etc. P = R (recent), F (first), or A (all).
CR x(m) = i,j,k...	Create a vector of length m and initialize its values to i, j, k, etc.
CR x(m) = i to j	Create a vector of length m and initialize its elements to values from i to j.
CR x(m,n) = i_{11},i_{12}, ...,i_{mn}	Create a table and initialize its values row by row to i_{11},...,i_{mn}. If no values are given the table is created but is not initialized.
IF (exp) command	Conditional execution of a command.
DO WHILE (exp) • • • end	Looping while "exp" is true.
DO FOR i = j,k • • • end	Looping

Figure 2—A sample of features of the development language

the important communications among these participants so that at each point in time it is clear what decisions have been made in the handling of each component of the system. SADF will store these communications in a network database and will relate them to the originator of the comment as well as to the component about which the comment is issued (see Figure 3.) This information is available to all participants in the development process and can be accessed by simple commands or queries.

In the course of system development, some of the necessary information for the formulation of system requirements can be captured from existing systems or existing documents, but the ultimate source of the information is the user. When developing a DSS, it is very unlikely, because of the newness of the field, that an old computer-based DSS will be in place before the development of a new one. Therefore, the user remains the only reliable source of information. However, different users have different needs and viewpoints that some-

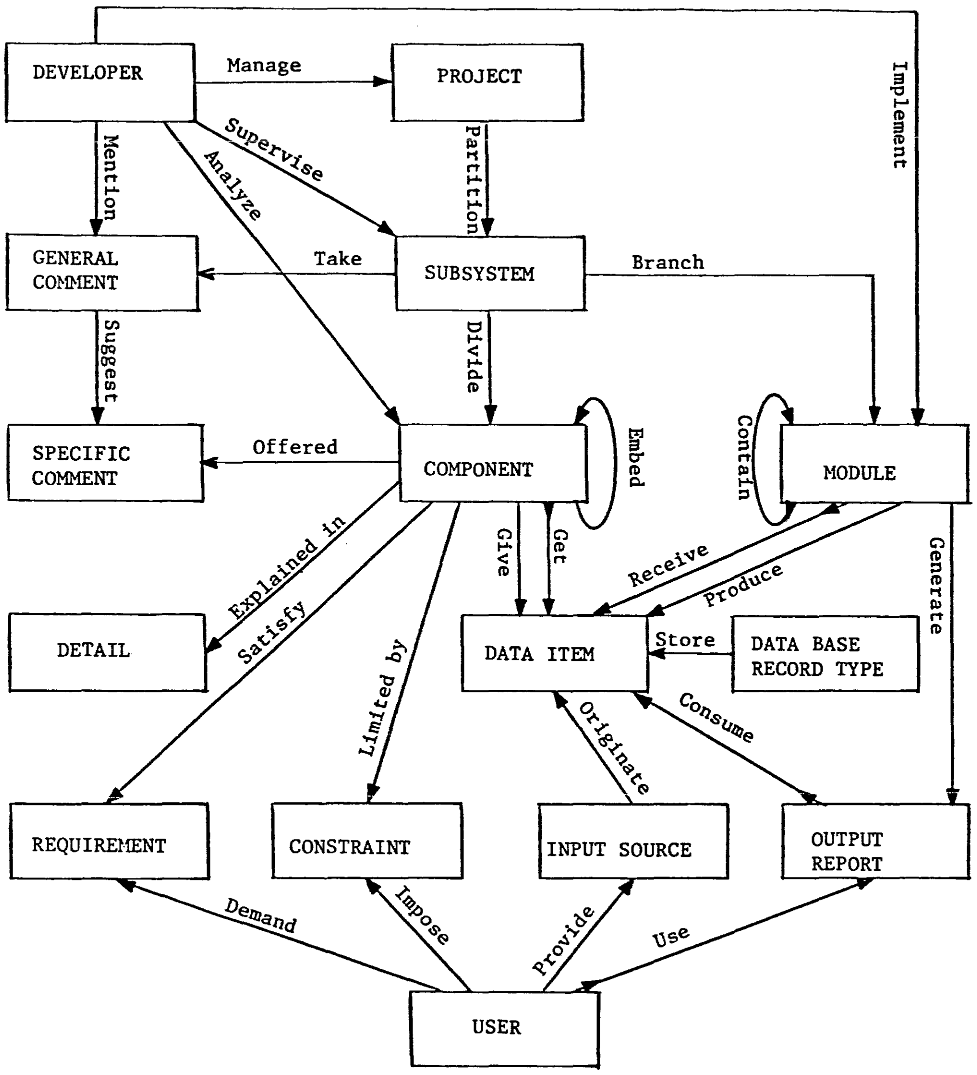


Figure 3—Extended network logical structure of the SADF database

times are in conflict. In any case, all viewpoints should be heard and all reasonable needs should be accounted for, according to some priority list. SADF stores this information in an extended network database[10] along with other information pertaining to the analysis and design of the DSS.

Part of the requirements could be obtained from the user through a program that would interview the user in a conversational mode through an interactive terminal. This could be easily accomplished by a questionnaire designed especially for solicitation of information from the user; however, instead of a human interviewer, the computer can be programmed to conduct the interview through an interactive terminal. Questions will be presented to the user, and answers will be obtained and stored in a database. After interviewing all users, a summary report will be produced and the results stored internally so that the report can be viewed by the analysts, designers, programmers, and so on. An automated interview usually is not sufficient to capture all requirements; however, it can help the analyst by revealing the problem areas requiring more extensive study. In any event, all obtained information will be stored in an extended network database (Figure 3). This method of storing the information facilitates the effective use of the information and provides an excellent means of documentation. A detailed discussion of SADF appears in Reference 2.

MODULE MANAGEMENT

One way to achieve high productivity in the process of software development is to use prefabricated pieces in the construction of a new system. The use of preprogrammed modules in the manufacturing of a new system not only increases the productivity of the software development process, but also increases the opportunity for producing high-quality software. Production of higher-quality software is possible in two ways:

First, the frequently used modules can be fine tuned to perform very efficiently. That is, these modules can be written in assembly language or they can be written by highly skilled programmers. Second, since preprogrammed modules presumably have been in use in other systems and environments they have been perfected. Also, the performance of these modules has been observed in actual practice, so their strengths and weaknesses are better known. The developer is therefore building his/her system with a better-known material so it is expected that a better system will be produced. In practice, however, the use of prefabricated pieces in the development of a new system is negligible, unless the same person is developing a similar system. The main reasons for not using the product of previous efforts in the development of a new system can be classified in the following categories:

1. Inflexible design—The module does not directly fit the current need, and inflexible design does not permit easy modification of the module.
2. Different programming language—The module is written in a different programming language with no interface to the language used for the system development.
3. Machine dependence—The module is written for a particular machine and cannot be used on other machines.
4. No organized information about the existence of the module exists—The modules are scattered in various places (e.g., files, tapes, computer cards, etc.). No one knows about their existence or there is no convenient way of getting information about them.
5. Lack of documentation—The existence of the module is known, but there is lack of documentation. The author is either unknown or is no longer with the organization, therefore, no one is sure how to use the module.
6. Lack of information about reliability of the module—The developer simply cannot trust someone else's product without having some evidence about the reliability of the product.
7. Lack of performance data about the module—There is no evidence to indicate how the module performs in practice.

If we are able to find a solution to these problems then we can expect to produce quality software with high efficiency and with reasonable cost.

The first problem calls for flexible design. Flexible design under the DSSDS is possible. Modification of a module through a source code manager (SCM) is also greatly facilitated. The second problem is solved under the DSSDS development language (DL) because DL provides interface to several programming languages, and any program written in DL can call modules of different programming languages. The third problem is less severe because most of the programs written in high-level languages are portable. There are several ways that this problem can be solved. If there is a complete program it can be routed to the right machine to be executed and the results transmitted to the originator of the problem. If the number of machine-dependent modules is considerable, a virtual machine (a simulation of another machine on an existing machine) can be developed to run these modules. Translators can also be written to translate programs of one given machine to another. The first solution is supported by DSSDS. The others can be designed and added to DSSDS if economically justifiable.

In this section we present a solution to problems 4 through 7. To solve these problems we need to create a centralized information base that contains all necessary information about all modules available to the software development center. This centralized information base can be used by individual members of the development team to select the appropriate modules to be incorporated in the development of the new systems.

To centralize all pertinent information about various modules, we design an extended network[10] database system, which we call the library of modules (LOM). We show how this library can be used to assist the developer in the task of module selection as well as to provide him/her with informative information in each problem area.

Our interest in preprogrammed modules is not stimulated only by productivity gains and production of quality software, but also because in the development of a DSS we need to supply the DSS with a collection of modules to be used by the problem processing system (PPS) and/or decision maker for model building activities. Therefore, we consider the library of modules an essential part of our DSSDS.

The Library of Modules (LOM)

The library of modules stores all desirable information about available modules in a centralized fashion. The developer after designing his/her system can turn to the module library and see which of the existing modules can be used in the development of the new system. If none of the modules can be used directly, the developer may then investigate if any of the modules can be used with minor modification. If the module library is large enough, it is reasonable to assume that some modules will be useful in the development of a new system. This will help a speedy development of a new system, which we consider an essential requirement for the development of decision support systems. This approach also provides an opportunity for developing high-quality software with reasonable cost.

The logical structure of an extended network database along with a proposed list of data items is shown in Figure 4. Modules are categorized by the problems they solve and the problems themselves are categorized by subject area. There may be more than one module for solving a given problem and a given module may solve more than one problem (*N:M* relationship). For each module, information about the name of the module, module number (similar to the call number for books), the purpose (what does it do?), technique used, and origin (where did it come from?) is stored. The record type THEORY is intended to represent the scientific basis of the technique used in the development of the module. The developer can check to see if there is a sound scientific basis for the technique, and if so, can educate him/herself and learn about the conditions under which the technique is valid. In other words THEORY does the job of a handbook. This can be very

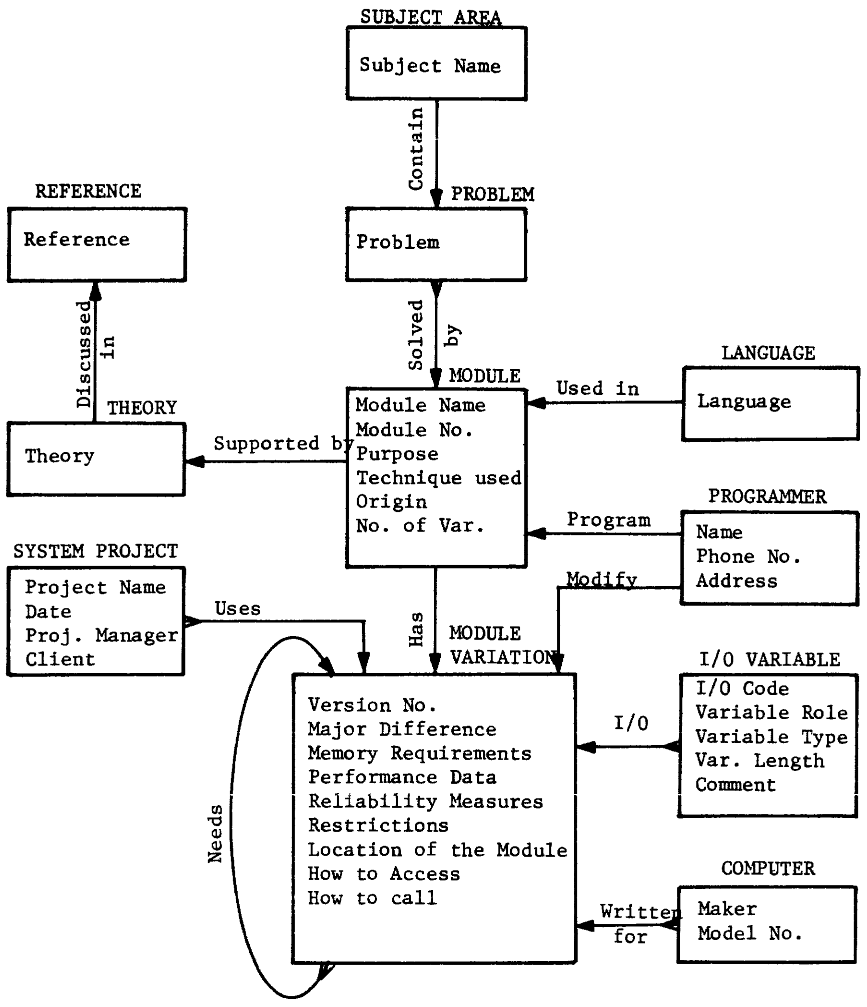


Figure 4—An extended network structure for the library of modules

helpful since the developer is not necessarily knowledgeable in all problem areas. For each THEORY a number of references are also given. Each module may have many variations (it is assumed that there is at least one variation, i.e., the original). Each record occurrence of the MODULE VARIATION record type contains the properties of a specific variation. These properties are shown in Figure 4. Major Difference is an explanation of the major difference between this version and original version of that module. Memory requirements gives the size of the program in bytes and is especially helpful when there is a memory restriction. Reliability and performance data essentially tell how reliable the module has been and how fast it runs. The other information includes restrictions of that variation, where it could be found, how it should be accessed, and what the calling procedure is. The record occurrence of each particular variation is associated with the system project(s) in which it has been used, and for each project the names of the project manager and client as well as the name of the project and date of development are given. So if the developer wants additional information about the development process or practical results he/she can contact the appropriate person. Each occurrence of record type LANGUAGE is related to all modules written in that particular language. So it is possible both to find out in what language a particular module is written and to scan through all modules written in a given language. Some variations of a module may be written for a particular computer so the record type COMPUTER is related to record type MODULE VARIATION through the many-to-many set, Written for. Each module variation is linked to its input/output through the set I/O. Properties of each I/O variable are stored in an occurrence of I/O VARIABLE. The I/O codes of I, O, or B correspond to input variables, output variables, and both respectively. Other data items of I/O variables are shown in Figure 4. Each module is linked to its programmer and each variation is linked to the programmer who did the modification. Each programmer's name, telephone number, and address is given so additional information can be obtained from the programmer if necessary.

Thus the library of modules (LOM), directly or indirectly (through references, addresses, etc.) includes all the information that the developer would like to know about a particular module. Another interesting feature of LOM is the set relationship Needs, which will be discussed in the next section.

Module Dependencies

Within a system it often happens that the output of a module is used as input by another module, thereby creating a dependency between the two modules. We call this dependency between two modules a context-sensitive association or a weak dependency, because the association is the result of input/output needs rather than the result of the direct need of one module for another. The dependency is context sensitive because it very much depends on the context; if module *x* in a given system needs the output of module *y* in order to work, in a different context (i.e., a different system), this may not be the case, because the input of *x* may be provided in another way (e.g., be simply read in).

In contrast to this dependency there is another kind of dependency, which we call strong dependency or a context free association. A strong dependency is the result of one module calling or invoking another module. For example if module *z* calls for the service of module *y* in its procedure, then we say *z* has a strong dependency on *y*, because *z* cannot function unless *y* is present. *y* in turn may have strong dependency on another model. In Figure 5, module *z* has strong

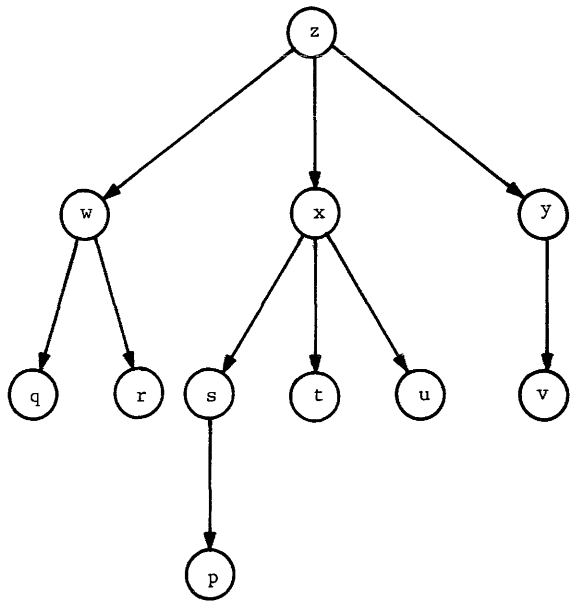


Figure 5—Strong dependencies among modules

dependency on *w, x,* and *y; w* in turn has a strong dependency on *q* and *r; x* is strongly dependent on *s, t,* and *u; s* in turn has strong dependency on *p;* and finally *y* is strongly dependent on *v*. These dependencies are context free because no matter in which system we use model *z*, it needs modules *w, x,* and *y* in order to operate. Modules *w, x,* and *y* in turn need the service of their own modules. This hierarchy continues until all the new modules stand alone and are self sufficient.

This strong dependency of one module on other modules is effectively captured by the recursive relation, Needs. This is an *N:M* relationship because each module may need the service of several other modules, and each module may give service to many modules. Treatment of module dependencies in this way greatly facilitates the development of new systems as well as modeling activities. Notice that if module *z* is selected to be included in a new system, all the modules that *z* is dependent on for service in a direct or indirect way should go with module *z*.

The linkage through set Needs provides valuable information to the developer. For example, if a module like *z* is a candidate for selection, the developer can scan through all other modules that are directly or indirectly needed by *z* and examine such properties as performance data, reliability measures, the language they are written in, hardware dependencies (if any), and so on. Examination of this information is important because it may reveal some undesirable properties of one or more modules in the collection, which may require the rewriting of those modules or the selection of an alternative module.

Another valuable benefit of this approach is that since through this linkage the developer can find out which modules use the service of a given module in a direct or indirect way, it is very easy to find out which modules will be affected by alteration of the given module, and therefore appropriate measures can be taken if necessary.

A third advantage of this approach is that it eliminates redundancy in the storage of modules. In other words, each module is stored only once, no matter how many other modules use its service.

Other consequences of this approach are that the system can evolve and can become personalized. The evolution is possible because new independent or dependent modules can be added to the system without difficulty. The developer can also use the original primitive modules and can build upon those a collection of modules to be used by him/herself on a personalized basis.

The system can also display a learning behavior. Observe that the set of modules that are directly needed by a module such as *z* can be considered as preconditions to *z*, because without them *z* cannot be executed. However, existence of a module in the database of the LOM automatically means that the preconditions are satisfiable, and in fact the linkage paths represent the solution paths to satisfy the preconditions. Any time a new problem is solved, that is, a new module is formulated with or without the use of existing modules, this new information is added to the system and the problem need not be solved again because the solution path to this problem already exists in the database. Thus the system displays a learning behavior. Moreover, these new skills are acquired in the area for which the system has been used and for which they are presumably needed the most. In other words, the system learns the right things. A final comment on the learning feature is in order: If the original collection of the primitive modules is considerably large, chances are that most of the new modules can be created through the use of the existing modules. It is the job of a human or computerized problem solver to combine the right ingredients to create a module that can deliver the desired results for a given task. It is expected that most new modules will result from combining the existing modules or from using some parts from the existing modules rather than from being created completely from scratch. Different schemes should result in different environments best suited to different lines of development.

Extensions to the Library of Modules

By making some conventions we can also add the information about the weak dependencies to the database. A weak dependency is the result of one module using the output of another module as its input. But inputs to a module generally can be provided by a variety of sources. For example, more than one module can provide input that can be used by a particular module. The inputs can also be read from a database, file, cards, and so on. So there are alternatives for the developer to choose from. The approach preferred depends on the kind of raw data available at a given context. It is beneficial to the developer if he/she is reminded of his/her choices. To include this new information we do not need to change the structure of our database but we need to make a few conventions. First we distinguish between three kinds of modules: a process module, which is a regular module and performs some data-processing task; an input module, which provides the inputs to a given module by reading them from the tape, from the database, from cards, and so on; and a link module, which links a process module to its alternative input modules. We let all three types of modules share the same record type; however each occurrence contains the information about the type of that module.

To help clarify this problem, let us consider an example. In Figure 6 module *z* needs modules *x and y and* some input that can be provided in three different ways. Either it can be provided by module *I1* by directly reading from some input source (e.g., from the database), *or* by module *I2* by directly reading from a different input source (e.g., from cards), *or* it can be provided as an output of module *w*. Module *w*, in order to work, needs module *v and* some input that can be provided in two alternative ways of *I3 or I4*. Notice that the *I* modules represent input modules and they are always terminal nodes in the dependency tree. The *L* modules are link modules and they always branch into alternative modules that can provide the input to the so-called owner module. Only one of the alternatives is necessary and sufficient to provide the input. The ordinary modules like *x*, *y*, and *w*, can call any of the other two types of modules or be self-sufficient.

Here the developer is provided with different alternatives for solving the problem although he/she may use the same module *z*. He/she may prefer one alternative over others in a given context or he/she may include some or all of the alternative solutions in the new system he/she builds and then let the user decide about a convenient approach in each problem situation.

Observe that Figure 6 closely resembles an AND/OR graph.[11,12] The process modules if they branch, represent AND or synthesis nodes, and the link modules represent the OR nodes. Since AND/OR graphs are used in the problem reduction approach to automated problem solving,[13] it follows that our database technique could be used as an effective mechanism in automatic problem solving. The complete AND/OR graph can be represented by the relationship Needs in the LOM database. Each node contains the information about whether it is an AND node or an OR node, and each linkage represents a reduction operator. Notice that the problem solving in this way is reduced to a search through the database. Moreover, if the start symbol (i.e., the module we are looking for) is found directly in the database, then the solution is guaranteed, provided the input data can be prepared in the right form. The system offers flexibility by allowing the input data to be fed to the module in various forms depending on the context. Different inputs may result in different combinations of modules that deliver the same results.

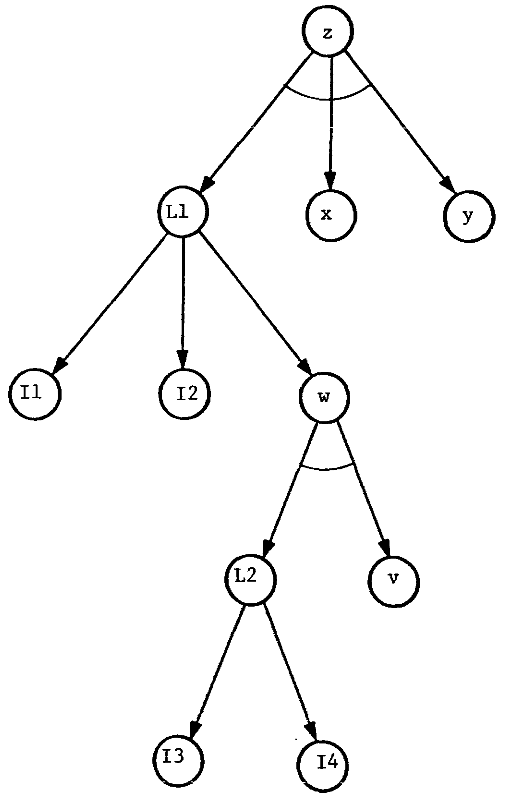
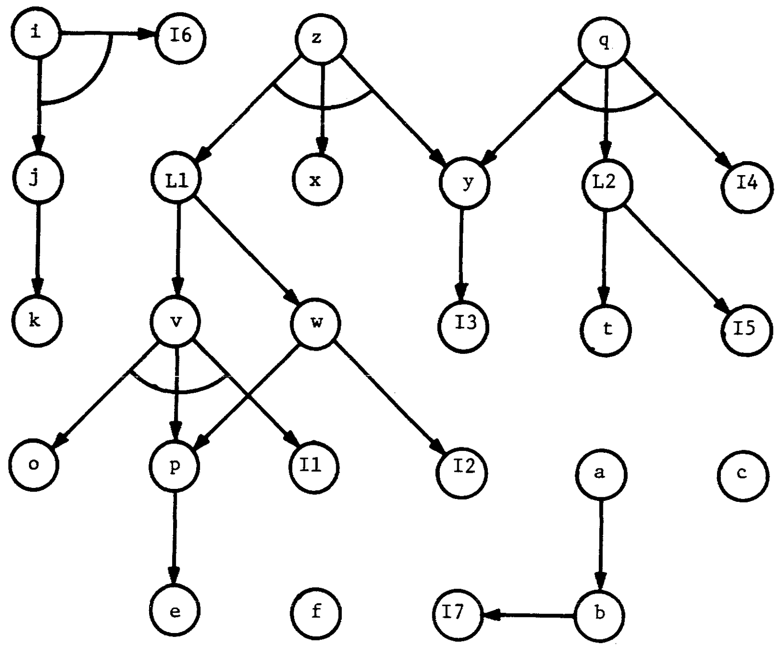


Figure 6—Strong and weak dependencies among modules

Alternatively, our module linkage mechanism can be the representation of a production system.[13] In other words, this mechanism can be used as a storage mechanism for a production system database (PSDB). If we consider our database as a representation of a production system, then all dependent modules are considered as nonterminals and the independent modules (I/O modules and self-sufficient modules) are considered as terminal nodes. Figure 7 shows the relationships of modules and their corresponding production system. Note: the collection of linkages emanating from a process module



```
1)  i → j, I6              7)  y → I3
2)  j → k                  8)  q → y | L2 | I4
3)  z → L1, x, y           9)  L2 → t | I5
4)  L1→ v | w             10)  a → b
5)  v → o, p, I1          11)  b → I7
6)  w → p |I2             12)  p → e
```

k, I6, O, e, f, I1, x, I7, I2, I3, t, I4, I5, and c are terminal nodes (i.e., stand alone modules).

Figure 7—Production system for modules in the library of modules

represents one production while each linkage emanating from a link module represents one production.

Module Management Language (MML)

The library of modules contains the information about any module accessible through the development environment. The source code of these modules may be stored in source-code files under the source code manager (SCM), or it may be stored in other files even under other computer systems. Regardless of the location of the module, all the information about its properties, location, and the procedure for accessing it is stored in the LOM. To use this information the developer needs a collection of tools so he/she can easily scan through the information in the library and select the desired modules. After the selection of the modules the developer wants to copy the module itself plus its supporting modules to an appropriate place to be included in the new system with or without some modifications.

The use of a network database management system for the library of modules automatically provides the developer with a powerful tool for retrieval and manipulation of information in the LOM. That is, the user can use the query language of DBMS and question the informational content of the database and/or manipulate the data. The developer can also develop a set of macrocommands that he/she can use repeatedly. Nevertheless, the existence of a module management language (MML) greatly facilitates the job of the developer. A set of basic commands is shown in Figure 8. Additional commands in the form of macros can be designed by the developer on a personalized basis and be added to the system. The MML is intended to be conversational in the sense that any ambiguities may be resolved through conversation with the user.

OTHER DEVELOPMENT FACILITIES

In the design of the foundation of DSSDS we implicitly assumed that a database management system (DBMS) exists. Moreover, we based our design on a network database system. Although it is possible to design a DSSDS without a database management system, existence of a DBMS greatly facilitates the design and implementation process. Besides,

ADD<rt>	to add a new occurrence of record type "rt" in the data base
DELETE<rt>	to delete an occurrence of record type "rt" from the data base
CHANGE<rt>	to change data item(s) within record type "rt". (The system prompts for additional information.)
DISPLAY<rt><x>	to display the informational content of occurrence x of "rt"
DISPLAY<rt>.<y>.<st>	to display the informational content of record type "rt" for all members (or owners) of owner (or member) y of set st
DISPLAY<rt>	to display all occurrences of rt. ("SYSTEM" is assumed to be the owner, otherwise, system prompts for the owner.)
DISPLAY OWNER<rt><x>.<st>	to display owner(s) of occurrence x of record type rt through set st
DISPLAY MEMBER<rt><x>.<st>	to display all members of occurrence x of record type rt through set st
DISPLAY SUBMODULE<x>.<it>	to display the value of item type "it" for all submodules directly needed by x, if "it" is missing all items will be displayed
DISPLAY ALL SUBMODULES<x>.<it>	to display the values of item type "it" for all direct or indirect submodules of x
DISPLAY SUPER MODULE<x>	to display modules that directly use the service of module x
DISPLAY ALL SUPER MODULES <x>	to display all modules that directly or indirectly use the service of module x
COPY<x>,<y>	to copy module x to file y
COPY ALL<x>,<y>	to copy x and all modules needed by x (directly or indirectly) to file y

Figure 8—A set of commands for a module management language

since the DSSDS normally would be used in a development center, existence of a DBMS in such a center is unquestionable. We also assumed the existence of a query language that would work with the database system.

In Figure 1 the existence of a report generator (RG), a graphics generator (GG), and a screen management language is recognized. We do not intend to discuss these facilities because these facilities do exist in a variety of forms. The report generator and graphics generator that we have in mind should have features similar to those of NOMAD,[14] for a screen management language we would like to have display facilities similar to those of SPF[15] or SCREEN MASTER.[16] The source code manager (SCM) is a tool that facilitates the generation of new modules or the alteration of existing modules. A detailed discussion of the SCM appears in Reference 2.

Request Handler (RH)

The request handler (RH) is intended to be used for maintenance purposes while the DSS is in operation. The purpose of the RH is to provide a communication link between the DSS and the DSSDS. The RH performs several important functions. First, suppose while the DSS is in operation, a bug is found in one of the modules. The user then sends a request through RH explaining the problem. The user does not necessarily know which programmer was involved in the development of that module. The RH by looking at the LOM can route the message to the right programmer. In case the programmer is unknown or is no longer with the organization, the RH will route the problem to the person in charge or the least-busiest person in charge of such problems.

Second, suppose that the user wants some extensions. That is, the user needs a new model that is not found in the DSS and that cannot be formulated through existing modules in the DSS by PPS or by the user him/herself. The RH will look at the LOM; if the module is found in the LOM, the RH will automatically access the module and route it to the DSS. Otherwise, the RH will place a message in the mail box of the least busiest developer or the developer with the right qualifications for that job. In case a user of the DSS has some questions and needs some help, he/she can send a help request to the RH. The request handler starts a dialogue with the user and gathers information about the subject and the nature of the question and then routes the message to an appropriate developer.

Through the RH the communication link between the DSSDS and the DSS remains open throughout the system's life cycle. Through this link the news about the availability of new modules, new versions of the existing modules, or new facilities can be sent to the DSS to be placed in the mail box of interested parties. RH provides a valuable facility for supporting the product during the operation phase of its life cycle.

DSS DEVELOPMENT

The DSSDS satisfies all the requirements we stated earlier for a DSS development system. That is, the DSSDS supports a speedy development of a DSS and it also supports the DSS in its entire life cycle. Various decision support systems can be developed through the DSSDS for different needs. The tools of the DSSDS are available independently, and finally, the DSSDS is capable of evolving over time.

With the initiation of a DSS project, systems analysis begins. SADF helps the developer to gather the information and store it in an organized way in a SADF database. Through a SADF all members of the development team can use the same data and share their thoughts. When the developer believes he/she understands the problem correctly, the development of the prototype begins. In prototyping the emphasis is on speedy development of a system that reasonably represents the proposed system. Through DSSDS a speedy development of a prototype is possible because the LOM can provide considerable preprogrammed modules. Besides, the modification of existing modules is greatly facilitated under the SCM. The Report Generator, the Graphics Generator and the query language are excellent facilities for prototyping, because efficiency is not an immediate concern in prototyping. For example if a special report has to be prepared, it is very likely that the report could be provided through RG quite easily. However if and when the report proved to be necessary and needed on a recurring basis then a new module should be written to create this report very efficiently for the final system.

SUMMARY

The need for many decision support systems in the near future stimulated our interest in finding a convenient way for developing such systems. The changing nature of DSS required us to find a way for speedy development and fast modification of DSS. Our study resulted in a proposal for a DSS development system (DSSDS). The DSSDS facilitates both the development and maintenance of a DSS. The philosophy of the DSSDS is based on two concepts: the use of highly automated tools throughout the development process and the use of prefabricated pieces in the manufacturing of a whole piece. The environment of the DSSDS consists of a development language (DL), a systems analysis and design facility (SADF), a module management language (MML), a source code manager (SCM), a report generator (RG), a graphics generator (GG), and a request handler (RH).

The DSSDS provides an environment in which the developer can create high-quality decision support systems, with moderate cost and in a relatively short period of time.

REFERENCES

1. Martin, J. *Application Development Without Programmers*. Englewood Cliffs, N.J.: Prentice-Hall, 1981.
2. Ghiaseddin, N.. "Framework for a DSS Development System," Ph.D. dissertation, Purdue University, 1982.
3. Boehm, R., "Software Engineering—R and D Trend and Defense Need." In *Research Directions in Software Technology*. Cambridge, Mass.: MIT Press, 1978.
4. Gomaa, H., and D. Scott. "Prototyping as a Tool in the Specification of User Requirements." *Proceedings of 5th International Conference on Software Engineering,* March 1981.
5. Teichrow, D., and E. Hershey. "PSL/PSA: A Computer Aided Technique for Structured Documentation and Analysis of Information Processing Systems." *IEEE Transactions on Software Engineering,* January 1977.
6. Bell, T., D. Bixler, and M. Dyer. "An Extendable Approach to Computer Aided Software Requirements Engineering." *IEEE Transactions on Software Engineering,* January 1977.
7. Ross, D., and W. Schomaman. "Structured Analysis for Requirements Definition." *IEEE Transactions on Software Engineering,* January 1977.
8. Stephens, S., and L. Tripp, "Requirements Expression and Verification Aid." *Proceedings of the 3rd International Conference on Software Engineering,* May 1978.
9. *Knowledge Manager Reference Manual,* Micro Data Base Systems, Inc., Lafayette, Indiana, 1983.
10. MDBS INC. *MDBS Application Programming Reference Manual.* Lafayette, Ind., 1981.
11. Nilsson, N., *Principles of Artificial Intelligence.* Palo Alto: Cal.: Tioga Publishing, 1980.
12. Bonczek, R., C. Holsapple, and A. Whinston, *Foundation of Decision Support Systems*. New York: Academic Press, 1981.
13. Davis, R., and J. King. "An Overview of Production Systems." In E. Elcock and O. Michie (eds.), *Machine Intelligence 8.* New York: Halsted Press, 1977.
14. McCracken, D.. *A Guide to NOMAD for Application Development,* Wilton, Conn.: National CSS, 1980.
15. Joslin, P., "System Productivity Facility," *IBM System Journal,* 20 (1981).
16. *SCREEN MASTER Reference Manual,* Micro Data Base Systems, Inc., Lafayette, Indiana, 1982

Applications of fuzzy languages and pictorial databases to decision support systems design

by EDWARD T. LEE
Memphis State University
Memphis, Tennessee

ABSTRACT

The pioneering work of D. T. Lee in developing an approach with major emphasis on database development has had a profound influence on the recent development of decision support systems, as well as on office information systems, database systems, and database machines. This database development approach is a new and powerful approach to decision support systems design methodologies.

In this paper the concepts of fuzzy languages and pictorial databases are applied to decision support systems design methodologies. First, fuzzy languages, fuzzy grammars, the classification of fuzzy grammars, derivation chain, degree of acceptance, and equivalence are defined. Operations like intersection, concatenation, Kleene closure, complement, and cardinality are also defined. Second, algebraic representation of the production system is presented and illustrated by examples. The difference between null string and the empty set of string is illustrated. Third, decision support systems involving geometric figures, chromosome images or leukocyte images are presented as illustrative examples.

In similarity retrieval from a pictorial database, very often it is desired to find pictures (or feature vectors, histograms, etc.) that are most similar to or most dissimilar to a test picture (or feature vector). Using similarity measures, one can not only store similar pictures logically or physically close to each other to improve retrieval or updating efficiency, one can also use such similarity measures to answer fuzzy queries involving nonexact retrieval conditions.

The applications of fuzzy languages and pictorial databases to decision support systems design methodologies offer what appears to be a fertile field for further study. The underlying ideas are interesting and easy for practical application. The results have useful applications in decision support systems, pattern recognition, pictorial information systems, and artificial intelligence.

A. INTRODUCTION

D. T. Lee's pioneering work in developing an approach that emphasizes database development[1–5] has profoundly influenced the recent development of decision support systems,[6–8, 24] as well as office information systems,[9] database systems,[10–11] and database machines. This database development approach is a new and powerful approach to designing decision support systems.

During the past several years, fuzzy languages[12] and similarity retrieval techniques[13] have attracted growing attention as promising avenues of approach to problems in decision support systems design methodologies.

B. FUZZY LANGUAGES

In the theory of formal languages, a language L is defined as a subset of V_T^*—as the set of strings over a finite alphabet V_T^*. Consequently, if x is a string in V_T^* then either x is a member of L or x is not a member of L. In contrast, in the case of a natural language, a sentence may be partially in a language, in the sense that it may be partially grammatically correct (or partially meaningful). Indeed, it may be argued that it is the complete precision of formal languages that sets them so sharply apart from natural languages.

The gap between formal and natural languages can be narrowed by introducing the concept of *fuzzy* language, that is, a fuzzy subset of V_T^*. In such a language, each string $x \in V_T^*$ is assigned a grade of membership, $\mu_L(x)$, in L, which for simplicity may be taken to be a number in the interval $[0, 1]$. When L is a natural language, $\mu_L(x)$ may be interpreted as an index of the grammatical correctness of x.

As was pointed out by E. T. Lee and Zadeh,[12] much of the existing theory of formal languages can be extended quite reaily to fuzzy languages. This has been done for the notions of grammar, Kleene closure, recursiveness, context-sensitive languages, context-free languages, regular languages, Chomsky normal form, Greibach normal form, and so on.[12] In the present paper, we shall focus our attention on the applications of fuzzy languages to decision support systems design methodologies. As in Lee and Zadeh, our notation, terminology and constructions for fuzzy languages will parallel closely those of Hopcroft and Ullman for nonfuzzy formal languages.[14]

We begin with a brief recapitulation of some of Lee and Zadeh's definitions.[12]

Let V_T be a set of terminals, with V_T^* denoting the set of strings over V_T. A fuzzy language L is a fuzzy subset of V_T^*. As such, it may be characterized by a membership function $\mu_L : V_T^* \longrightarrow [0, 1]$ that associates with each string x in V_T^* its grade of membership, $\mu_L(x)$, in L.

If L_1 and L_2 are two fuzzy languages, then their *union* is a fuzzy language denoted by $L_1 + L_2$ and defined by

$$\mu_{L_1+L_2} = \max[\mu_{L_1}(x), \mu_{L_2}(x)], \; x \in V_T^*, \tag{1}$$

or more compactly

$$\mu_{L_1+L_2} = \mu_{L_1} \vee \mu_{L_2}, \tag{2}$$

where $\vee$ stands for Max (on infix form). To simplify the notation, we shall write throughout $\vee$ for Max, $\wedge$ for Min and will omit the arguments of membership functions when an equality or inequality that these functions satisfy holds for all strings in V_T^*.

The *intersection* of L_1 and L_2 is denoted by $L_1 \cap L_2$ and is defined by

$$\mu_{L_1 \cap L_2} = \mu_{L_1} \wedge \mu_{L_2}. \tag{3}$$

The *concatenation* of L_1 and L_2 is denoted by L_1L_2 and is defined by

$$\mu_{L_1L_2}(x) = \bigvee_u (\mu_{L_1}(u) \wedge \mu_{L_2}(x-u)), \qquad x \in V_T^*, \tag{4}$$

where u and $x - u$ denote, respectively, a prefix and the corresponding suffix of x.

The *Kleene closure* of a language L is defined by

$$L^* = \xi + L + LL + LLL + \ldots \tag{5}$$

where ξ denotes the null string.

The *complement of a language* L is denoted by $\overline{L}$ and is defined by

$$\mu_{\overline{L}} = 1 - \mu_L. \tag{6}$$

The *cardinality* of a language L is denoted by

$$|L| = \sum_i \mu_L(x_i). \tag{7}$$

A *fuzzy grammar* is a quadruple $G = (V_N, V_T, P, S)$ where V_N is a set of non-terminals (i.e., labels for certain fuzzy subsets of L), P is a set of fuzzy productions and $S \in V_N$. A generic fuzzy production has the form

$$\alpha \xrightarrow{\rho} \beta, \tag{8}$$

where $\alpha, \beta \in (V_T \cup V_N)^*$ and $0 < \rho \leq 1$.

If $\alpha_1, \ldots, \alpha_m$ are strings in $(V_T \cup V_N)^*$ and

$$\alpha_1 \xrightarrow{\rho_2} \alpha_2, \ldots,$$
$$\alpha_{m-1} \xrightarrow{\rho_m} \alpha_m,$$

then α_m is *derivable* from α_1 in grammar G. This is expressed as

$$\alpha_1 \underset{G}{\Rightarrow} \alpha_m$$

or simply $\alpha_1 \Rightarrow \alpha_m$. The expression

$$\alpha_1 \xrightarrow{\rho_2} \alpha_2 \ldots \alpha_{m-1} \xrightarrow{\rho_m} \alpha_m \tag{9}$$

is called a *derivation chain from* α_1 to α_m. The strength of such a chain is defined to be $\rho_2 \wedge \ldots \wedge \rho_m$, which may be interpreted as the strength of its weakest link.

A fuzzy grammar G generates a fuzzy language $L(G)$ in the following manner. A string of terminals x is in $L(G)$ if and only if x is derivable from S. The grade of membership of x in $L(G)$ is given by

$$\mu_G(x) = \sup(\rho_1 \wedge \ldots \wedge \rho_{m+1}) \tag{10}$$

where $S \xrightarrow{\rho_1} \alpha_1 \ldots \alpha_m \xrightarrow{\rho_{m+1}} x$ is a derivation chain from S to x and the supremum is taken over all derivation chains from $S \in x$. In words,

$$\mu_G(x) = \text{strength of the strongest derivation chain from } S \text{ to } x.$$

Thus, (10) defines $L(G)$ as a fuzzy set in $(V_T \cup V_N)^*$. If $L(G_1) = L(G_2)$ in the sense of equality of fuzzy sets, then the grammars G_1 and G_2 are said to be *equivalent*.

This concludes the recapitulation of some of the basic concepts defined by Lee and Zadeh.[12]

C. ALGEBRAIC REPRESENTATION OF THE PRODUCTION SYSTEM

For many purposes it is convenient to replace P by a set of algebraic equations involving the operations of concatenation and addition (union) of fuzzy sets of strings. Such equations constitute an extension to fuzzy sets of strings of the approach used by Rosenkrantz to derive the Greibach normal form.[15]

The replacement is effected as follows. If P contains productions of the form

$$\alpha \xrightarrow{\rho_1} \beta, \quad \alpha, \beta \in (V_T \cup V_N)^*$$
$$\alpha \xrightarrow{\rho_2} \gamma$$

then

$$\alpha = \rho_1\beta + \rho_2\gamma, \tag{11}$$

where + denotes the union of the fuzzy sets of strings represented by $\rho_1\beta$ and $\rho_2\gamma$.

Example 1. Written in algebraic form, the production system of

$$\begin{array}{ll} S \xrightarrow{0.5} AB & A \xrightarrow{0.6} b \\ S \xrightarrow{0.8} A & B \xrightarrow{0.4} A \\ S \xrightarrow{0.8} B & B \xrightarrow{0.2} a \\ A \xrightarrow{0.5} a & AB \xrightarrow{0.4} BA \end{array}$$

reads

$$\begin{aligned} S &= 0.5AB + 0.8A + 0.8B \\ A &= 0.5a + 0.6b \\ B &= 0.4A + 0.2a \\ AB &= 0.4BA \end{aligned}$$

As in the case of nonfuzzy languages, it is convenient to classify the grammars of fuzzy languages into four principal categories, which follow, in order of decreasing generality.

C.1 Type 0 Grammars

In this case, productions are of the general form

$$\alpha \xrightarrow{\rho} \beta,$$

where α and β are strings in $(V_T + V_N)^*$, with $\alpha \neq \xi$.

C.2 Type 1 Grammars (Context Sensitive)

Here the productions are of the form

$$\alpha \xrightarrow{\rho} \beta,$$

where α and β are strings in $(V_T + V_N)^*$, with $\alpha \neq \xi$ and $|\beta| \geq |\alpha|$, that is, the length of the right-hand side (the consequent) must be at least as great as the length of the left-hand side (the antecedent).

C.3 Type 2 Grammars (Context Free)

Here the allowable productions are of the form

$$A \xrightarrow{\rho} \alpha$$

where $A \in V_N$, $\alpha \in (V_T + V_N)^*$, and $S \xrightarrow{\rho} \xi$ is allowed. Thus, in the case of a context-free grammar, A can be replaced by α regardless of the context in which A occurs.

C.4 Type 3 Grammars (Regular)

In this case the allowable productions are of the form

$$A \xrightarrow{\rho} aB$$
$$A \xrightarrow{\rho} a$$
$$S \xrightarrow{\rho} \xi,$$

where A, $B \in V_N$ and $a \in V_T$.

In solving a system of algebraic equations representing the production system of a fuzzy grammar, one frequently encounters linear equations of the form

$$X_1 = P_1 X_1 + Q_1, \tag{12}$$

in which X_1, P_1, and Z_1 are fuzzy sets of strings over a finite alphabet, and + and the product denote the union and the concatenation, respectively.

Let $L_\xi = \{\xi\}$ = null string;

$$LL_\xi = L_\xi L = L. \tag{13}$$

L_ξ plays the role of the unit element for concatenation.

Let θ = the empty set of strings, then

$$A\theta = \theta A = \theta \tag{14}$$

since there are no elements in θ.

Proposition 1.[16] If P_1 does not contain the null string, then (12) has a unique solution for X_1, which is given by

$$X_1 = P_1^* Q_1, \tag{15}$$

where P_1^* is the Kleene closure of P_1 (in the sense of (5)).

The algebraic notation that was described earlier is particularly useful in the case of context-free grammars. Thus if the nonterminals in V_N are denoted by $X_1, \dots, X_n$, and $X = (X_1, \dots, X_n)$, with $X_1 = S$, then the production system P can be put into the form

$$X = f(X) \tag{16}$$

where f is an n-vector whose components are multinomials in the X_i, $i = 1, \dots, n$. In this way, the determination of the fuzzy set of strings generated by the grammar reduces to finding a fixed point of the function f. In this connection, it can really be shown that if we set $X^0 = \theta$ = empty set and form the iterates

$$X^{k+1} = f(X^k), X^0 = \theta, \qquad k = 1, 2, 3, \dots, \tag{17}$$

then, for each k, X^k is a fuzzy subset of the solution of (16).

We are now ready to turn our attention to the applications of fuzzy languages to decision support systems design methodologies. Algorithms for finding Chomsky and Greibach normal forms for a fuzzy context-free grammar using an algebraic approach have been found.[17]

D. DECISION SUPPORT SYSTEMS INVOLVING GEOMETRIC FIGURES

Pictures, images, and figures play basic and important roles in decision support systems.

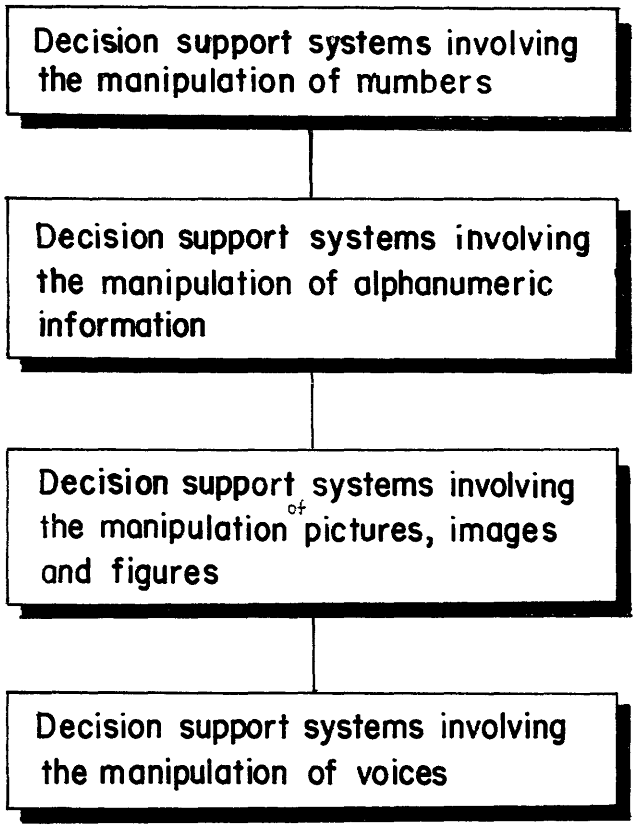

Decision support systems involving the manipulation of numbers
Decision support systems involving the manipulation of alphanumeric information
Decision support systems involving the manipulation of pictures, images and figures
Decision support systems involving the manipulation of voices

Figure 1—A brief summary of the history of decision support systems

In examining the history of the development of decision support systems, we discover that early decision support systems were developed primarily for applications related to scientific computation, as in weather prediction, aerospace applications, and nuclear physics applications. At this stage, the decision support system served as a big calculator to perform, in the main, scientific computation. Then it was found that decision support systems could also be used for business applications, information storage and retrieval, word processing, and report generation.

New frontiers in designing decision support systems are the representation, processing, storage, and retrieval of pictures. A brief summary of the history of decision support systems is shown in Figure 1.

Types of pictorial information amenable to decision support systems include geometric figures, chromosome and leukocyte images, maps, fingerprints, and human faces. Work is also being done in the areas of computerized tomography and the interpretation of earth-resource satellite photographs (e.g., to identify the rivers and highways in a particular area and to discriminate vegetation types).

A classification of pictures is shown in Figure 2. Thus, we select triangles to be worked on first.

Given a triangle $\triangle ABC$ with angles A, B, and C, quantitative measures of the similarity of this triangle to isosceles triangles, equilateral triangles, and right triangles may be defined as

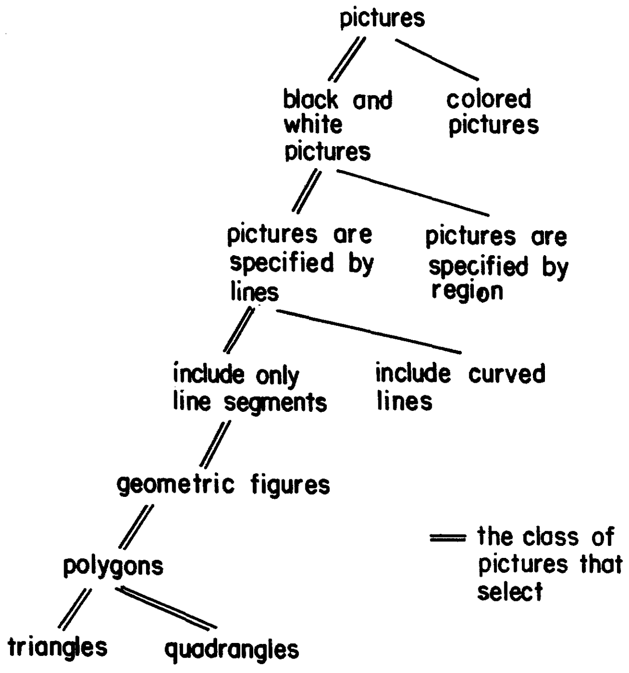

pictures
black and white pictures
colored pictures
pictures are specified by lines
pictures are specified by region
include only line segments
include curved lines
geometric figures
polygons
triangles
quadrangles
= the class of pictures that select

Figure 2—A classification of pictures

$$\mu_I(\triangle ABC) = 1 - \frac{1}{60^\circ}\min\{|A-B|, |B-C|, |C-A|\},$$

$$\mu_E(\triangle ABC) = 1 - \frac{1}{180^\circ}\max\{|A-B|, |B-C|, |C-A|\},$$

and

$$\mu_R(\triangle ABC) = 1 - \frac{1}{90^\circ}\min\{|A-90^\circ|, |B-90^\circ|, |C-90^\circ|\},$$

respectively.

A quantitative measure of the similarity of $\triangle ABC$ to isosceles right triangles may be defined as

$$\mu_{IR} = \min\{\mu_I, \mu_R\}$$

or

$$\mu'_{IR} = \mu_I \cdot \mu_R.$$

Since both μ_I and μ_R are in the range 0 to 1,[18] $\mu_{IR} \geq \mu'_{IR}$. Depending on its prospective application, μ_{IR} may be substituted for μ'_{IR} or vice-versa.

A quantitative measure of the similarity of $\triangle ABC$ to scalene triangles may be defined as:

$$\mu_{SC} = 1 - \max\{\mu_I, \mu_R, \mu_E\}.$$

For a triangle $\triangle ABC$ with $A \geq B \geq C$, we shall use the following vector representation for convenience:

$$\triangle ABC = (A, B, C).$$

Lemma 1. Given a triangle $\triangle ABC$ with angles A, B, C, if we assume that $A \geq B \geq C$, then

$$\mu_I(\triangle ABC) = 1 - \frac{1}{60^\circ}\min\{A-B, B-C\}$$
$$\mu_E(\triangle ABC) = 1 - \frac{A-C}{180^\circ}$$
$$\mu_R(\triangle ABC) = 1 - \frac{|A-90^\circ|}{90^\circ}.$$

Lemma 2. Given a triangle $\triangle ABC$, the set $\{0, \mu'_{IR}(\triangle ABC), \mu_{IR}(\triangle ABC), \mu_I(\triangle ABC), \mu_R(\triangle ABC), 1\}$ with max and min as the two binary operations forms a distributive but not complemented lattice.

Example 2. Table I gives the μ_I, μ_E, μ_R, μ_{IR}, and μ_{SC} of 12 triangles.

In the paper by Zadeh,[19] "not," "and," and "or" are interpreted as "the operation of complementation (or, equivalently, negation)," "the operation of intersection," and "the operation of union," respectively. In another paper by Zadeh,[20] such linguistic hedges as "very," or "more or less," were viewed as operators that act on the fuzzy sets representing the meaning of their operands. More specifically, "very" is interpreted as "the operation of concentration," which has the effect of squaring the membership function; "more or less" is interpreted as "the operation of dilation," which has the effect of taking the square root of the membership function. Composite fuzzy queries can be answered by using linguistic hedges and quantitative fuzzy semantics.

Example 3. The grade of membership of the triangle (90°, 75°, 15°) with respect to the class "very similar to isosceles triangles" is 9/16; its grade of membership in the class "more or less similar to isosceles triangles" is $\sqrt{3}/2$.

Example 4. Assuming the 12 triangles in Table I as the database, the composite fuzzy query, "retrieve the triangles that are very similar to equilateral triangles and more or less similar to right triangles" may be answered by computing

TABLE I—Similarity measures for 12 triangles, $\triangle ABC$

$\triangle ABC$	μ_I	μ_E	μ_R	μ_{IR}	μ_{SC}
1. (90°, 70°, 20°)	⅔	11/18	1	⅔	0
2. (90°, 60°, 30°)	½	⅔	1	½	0
3. (120°, 60°, 0°)	0	⅓	⅔	0	⅓
4. (60°, 60°, 60°)	1	1	⅔	⅔	0
5. (90°, 45°, 45°)	1	¾	1	1	0
6. (180°, 0°, 0°)	1	0	0	0	0
7. (75°, 60°, 45°)	¾	⅚	⅚	¾	⅙
8. (75°, 75°, 30°)	1	¾	⅚	⅚	0
9. (90°, 75°, 15°)	¾	7/12	1	¾	0
10. (120°, 30°, 30°)	1	½	⅔	⅔	0
11. (120°, 45°, 15°)	½	5/12	⅔	½	⅓
12. (150°, 15°, 15°)	1	¼	⅓	⅓	0

TABLE II—Similarity measures with linguistic hedges for the 12 triangles of Table I: "Very similar to an equilateral triangle," "more or less similar to a right triangle," and "both very similar to an equilateral triangle and more or less similar to a right triangle"

$\triangle ABC$	μ_E^2	$\mu_R^{1/2}$	$\mu_E^2 \wedge \mu_R^{1/2}$
1. (90°, 70°, 20°)	0.37	1	0.37
2. (90°, 60°, 30°)	0.44	1	0.44
3. (120°, 60°, 0°)	0.11	0.82	0.11
4. (60°, 60°, 60°)	1	0.82	0.82
5. (90°, 45°, 45°)	0.56	1	0.56
6. (180°, 0°, 0°)	0	0	0
7. (75°, 60°, 45°)	0.69	0.91	0.69
8. (75°, 75°, 30°)	0.56	0.91	0.56
9. (90°, 75°, 15°)	0.34	1	0.34
10. (120°, 30°, 30°)	0.25	0.82	0.25
11. (120°, 45°, 15°)	0.18	0.82	0.18
12. (150°, 15°, 15°)	0.06	0.53	0.06

1. $\mu_E{}^2$, the membership function, very similar to equilateral triangles
2. $\mu_R{}^{1/2}$, the membership function, more or less similar to right triangles
3. min $[\mu_E{}^2, \mu_R{}^{1/2}]$, *denoted by* $\mu_E{}^2 \wedge \mu_R{}^{1/2}$, the membership function very similar to equilateral triangles and more or less similar to right triangles

These are given in Table II.

If we set the threshold to be 0.6, then the answer to the query is triangles (60°, 60°, 60°) and (75°, 60°, 45°). Absence of a threshold is interpreted as a threshold of 0; any element with a grade of membership greater than 0 will be part of the answer; then the answer to our fuzzy query is the fuzzy set $\{(\triangle ABC), \mu_E{}^2(\triangle ABC) \wedge \mu_R{}^{1/2}(\triangle ABC)\}$, where $\mu_E{}^2 \wedge \mu_R{}^{1/2}$ is the membership function.

Zadeh discusses the classification of linguistic hedges and the operations of contrast intensification, fuzzification, and accentuation.[20]

Elsewhere,[18] for $\triangle x$ with angles $A \geq B \geq C$ and $\triangle y$ with angles $A' \geq B' \geq C'$, we have set the similarity between $\triangle x$ and $\triangle y$ equal to

$$\mu_A(\triangle x, \triangle y) = 1 - \frac{1}{240°}\{ |A - A'| + |B - B'| + |C - C'| \}.$$

Example 5. Given a triangle $\triangle y$ with angles $A' \geq B' \geq C'$ and a tolerance ϵ with $0 \leq \epsilon \leq 1$, the fuzzy query "retrieve all the triangles that are similar to $\triangle y$ within a tolerance of ϵ" may be carried out as follows:

Let X be the set of triangles $\triangle x$ in the database with angles $A \geq B \geq C$.

Since angles A, B, C are ordered from greatest to least, the ranges for the angles must be

$$60° \leq A \leq 180°,$$

$$0° \leq B \leq 90°,$$

$$0° \leq C \leq 60°.$$

Instead of testing each triangle $\triangle x$ in X to see if $\mu_A(\triangle x, \triangle y) \geq 1 - \epsilon$, first form the subset X' of possible candidates by using one of the following three methods:

1. Find the set of triangles $\triangle x'$ with A in the range of $\max\{60°, A' - 240°\epsilon\} \leq A \leq \min\{180°, A' + 240°\epsilon\}$.
2. Find the set of triangles $\triangle x'$ with B in the range of $\max\{0°, B' - 240°\epsilon\} \leq B \leq \min\{90°, B' + 240°\epsilon\}$.
3. Find the set of triangles $\triangle x'$ with C in the range of $\max\{0°, C' - 240°\epsilon\} \leq C \leq \min\{60°, C' + 240°\epsilon\}$.

Those triangles $\triangle x'$ in X' where $\mu_A(\triangle x', \triangle y) \geq 1 - \epsilon$ satisfy the query.

Example 6. Let triangle $\triangle y$ have angles $A' = 100°$, $B' = 60°$, and $C' = 20°$, and let $\epsilon = .20$. By using method 1, the set of possible candidates for the fuzzy query "retrieve all the triangles that are similar to $\triangle y$ within a tolerance of ϵ" consists of those triangles where

$$\max\{60°, 52°\} \leq A \leq \min\{180°, 148°\}$$

Thus, we obtain

$$60° \leq A \leq 148°,$$

and

$$\mu_A(\triangle x', \triangle y) \geq .8.$$

If $\epsilon = .10$ then the set of possible candidates becomes those triangles where $60° \leq A \leq 124°$.

A dissimilarity measure of $\triangle x$ and $\triangle y$ may be defined as the complement of the similarity of $\triangle x$ and $\triangle y$:

$$\mu_{\mathrm{DIS}}(\triangle x, \triangle y) = 1 - \mu_A(\triangle x, \triangle y)$$

Example 7. Given two triangles $\triangle x$ and $\triangle y$, the fuzzy query "retrieve all the triangles that are more or less dissimilar to $\triangle x$ or very very similar to $\triangle y$," may be answered by computing $\mu_{\mathrm{DIS}}^{1/2}(\triangle t, \triangle x) \vee \mu_A^4(\triangle t, \triangle y)$ for all triangles $\triangle t$ in the database, where $\vee$ is an infix operator for max.

Therefore, other composite fuzzy queries involving chromosomes or leukocytes can also be answered in the same manner. Examples of composite fuzzy queries involving chromosomes are "retrieve the chromosomes that are very similar to median chromosomes, but not similar to a given chromosome," and "retrieve the chromosomes that are more or less similar to median chromosomes and very very similar to a given chromosome." Examples of composite fuzzy queries involving leukocytes are "retrieve leukocytes with deeply indented nucleus or round nucleus" and "retrieve leukocytes with elongated nucleus or slightly indented nucleus."

In what follows, propositions are presented for an effective way of organizing a shape-oriented triangle, chromosome, or leukocyte database.

Proposition 2. For shape-oriented storage of triangles, it is advantageous to store the angles of a triangle in decreasing order of magnitude. This representation may be viewed as a normal form for shape-oriented triangle representation.

Proposition 3. For shape-oriented storage of triangles, if we logically order all the triangles individually and independently according to the magnitude of the angles A, B, and C, then we can reduce retrieval time for answering queries.

Proposition 4. For shape-oriented storage of triangles, we can use associative memory with match-on-between-limits and find-match-count operations[21] or other hardware searching facilities such as the Symbol-2R computer[22] to reduce retrieval time.

E. DECISION SUPPORT SYSTEM INVOLVING CHROMOSOME IMAGES

A preliminary study of applying shape-oriented similarity measures defined over a pair of chromosome images to the classification problem has been presented.[23] In this paper, the classification problem is studied through the use of shape-oriented measures of the similarity of a given chromosome image to symmetrical chromosomes, median chromosomes, submedian chromosomes, and acrocentric chromosomes.

The best-fit skeletal "length and angle only" transformation with angles A_i and sides a_j is shown in Figure 3. After connecting the tips of arms no. 1 and 2, and the tips of arms no. 3 and 4, as indicated by dotted lines, the best-fit skeletal "length and angle only" transformation becomes a hexagon.

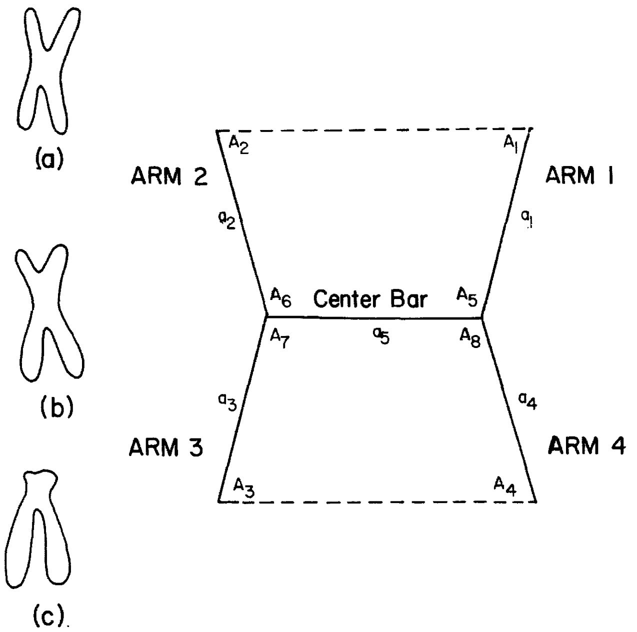


Figure 3—Chromosome images: (a) median, (b) submedian, (c) acrocentric, (d) best-fit skeletal "length and angle only" transformation

E.2 Symmetry of Chromosome Images

The preparation of chromosome images and the definition of metaphase chromosome images is discussed by Widrow.[25] At metaphase each chromosome has a twin, normally identical counterpart.

Definition 1. A chromosome image with angles A_i and sides a_j is a *symmetrical* chromosome image if and only if $A_{2i-1} = A_{2i}$ for $1 \le i \le 4$, $a_1 = a_2$ and $a_3 = a_4$.

A shape-oriented quantitative measure of the similarity of a given chromosome image A to all symmetrical chromosome images may be defined as

$$\mu_s(A) = 1 - \rho_s \sum_{i=1}^{4} |A_{2i-1} - A_{2i}|$$

where ρ_s is a normalization constant to be determined.

Lemma 3. A chromosome image is a symmetrical chromosome image if and only if $\mu_s = 1$.

In what follows, we assume that

$$A_i \le 180° \text{ for } 1 \le i \le 8.$$

The angles of chromosome image A are represented in a vector form as

$$A = (A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8).$$

E.2 Determination of the Normalization Constant ρ_s

The purpose of ρ_s is to normalize the value of ρ_s between 0 and 1. In order to determine the value of ρ_s, we must first determine $\max\{\sum_{i=1}^{4} |A_{2i-1} - A_{2i}|\}$. After we find this value, we assume μ_s to be equal to zero at this extreme case, so that we can determine the value of ρ_s.

Lemma 4. For any chromosome image A with angles A_i,

$$\max_{A} \left\{ \sum_{i=1}^{4} |A_{2i-1} - A_{2i}| \right\} = 720°.$$

Definition 2. A chromosome image is a *most unsymmetrical* chromosome image if and only if $\mu_s = 0$.

Theorem 1

$$\rho_s = \frac{1}{720°}$$

Proof. Assuming that the symmetry measure (μ_s) of a most unsymmetrical chromosome image is equal to zero, then

$$\mu_s = 0 = 1 - \rho_s \cdot 720° \text{ and } \rho_s = \frac{1}{720°}$$

Corollary 1. Given a chromosome image A with angles A_i,

$$\mu_s(A) = 1 - \frac{1}{720°} \sum_{i=1}^{4} |A_{2i=1} - A_{2i}|.$$

E.3 Most Unsymmetrical Chromosome Images

Shape-oriented quantitative measures of the dissimilarities of a given triangle to isosceles triangles, right triangles, and equilateral triangles have been defined.[23] According to the measures defined, it was also proved that the triangle most dissimilar to an isosceles triangle is (120°, 60°, 0°); the triangle most dissimilar to an equilateral triangle is (180°, 0°, 0°); and the triangle most dissimilar to a right triangle is (180°, 0°, 0°). It is of interest to ask whether the most unsymmetrical chromosome image is unique or not. If not, then what are all the most unsymmetrical chromosome images? In order to answer this question in a concise form, we need to define an equivalence relation among chromosome images. We first define the following four permutations of the angles

$$\begin{aligned} P_0(A) &= A \\ P_T(A) &= (A_2, A_1, A_4, A_3, A_6, A_5, A_8, A_7) \\ P_R(A) &= (A_4, A_3, A_2, A_1, A_8, A_7, A_6, A_5) \\ P_{TR}(A) &= (A_3, A_4, A_1, A_2, A_7, A_8, A_5, A_6). \end{aligned}$$

P_0 is the identical permutation. P_T is the permutation that interchanges the right-hand-side angles and the left-hand-side angles. P_R is the permutation that interchanges the upper angles and the lower angles. P_{TR} is the composite permutation of P_T and P_R. P_{TR} may also be interpreted as the rotation of a chromosome image 180° about an axis perpendicular to its own plane.

Denote these four permutations as the set G,

$$G = \{P_0, P_T, P_R, P_{TR}\}.$$

Lemma 5. Given a chromosome image A, μ_s is invariant over G.

$$\mu_s(A) = \mu_s(g(A)), \qquad g \in G.$$

Lemma 6. The set G over A forms an Abelian group.

Definition 3. Given two chromosome images A and B, A is *angularly equal* to B denoted as $A = B$ if and only if $A_i = B_i$ for $1 \le i \le 8$.

Definition 4. Given two chromosome images A and B, A is *angularly equivalent* to B denoted as $A \equiv B$ if and only if $A = g(B)$ for some g in G.

Example 8. Given chromosome images A and B with

$$A = (180°, 0°, 180°, 0°, 180°, 0°, 180°, 0°)$$
$$B = (0°, 180°, 0°, 180°, 0°, 180°, 0°, 180°)$$

then $A \equiv B$.

Theorem 2. There are six most unsymmetrical chromosome images, namely

$$\begin{aligned} A^1 &= (180°, 0°, 180°, 0°, 180°, 0°, 180°, 0°) \\ A^2 &= (180°, 0°, 180°, 0°, 180°, 0°, 0°, 180°) \\ A^3 &= (180°, 0°, 180°, 0°, 0°, 180°, 0°, 180°) \\ A^4 &= (180°, 0°, 0°, 180°, 180°, 0°, 180°, 0°) \\ A^5 &= (180°, 0°, 0°, 180°, 0°, 180°, 180°, 0°) \\ A^6 &= (180°, 0°, 0°, 180°, 180°, 0°, 0°, 180°). \end{aligned}$$

Proof. $\mu_s(A) = 0$ implies that $\sum_{i=1}^{4} |A_{2i-1} - A_{2i}| = 720°$. This implies that

$$|A_{2i-1} - A_{2i}| = 180°, \qquad 1 \le i \le 4.$$

Therefore, either

$$A_{2i-1} = 0° \text{ and } A_{2i} = 180°$$

or

$$A_{2i-1} = 180° \text{ and } A_{2i} = 0°, \text{ for } 1 \le i \le 4.$$

Thus, there are 16 most unsymmetrical chromosome images not angularly equal to each other. By using the equivalence relation defined in Definition 4, these 16 most unsymmetrical chromosome images are degenerated into six equivalent classes, and A^1, A^2, A^3, A^4, A^5, and A^6 are the representatives of these six equivalent classes.□

E.4 Median Images

When classifying chromosome images, different types and sizes are encountered. Three of these different types, median, submedian, and acrocentric will be examined. They all have the same general appearance. The difference occurs in the location of their centromere. Both figures and string representations of median, submedian, and acrocentric chromosome images are given by Lee and Fu.[26] String representations were done by using the terminal set $V_T = \{a, \swarrow; b, \nwarrow; c, \downarrow; d, \nearrow\}$. The same terminal set was used by Ledley et al.[27] String representation of a median chromosome image is as follows:

$$\chi_{(\text{median})} = cbbbabbbbdbbbbabbbcbbbabbbbdbbbbabbb.$$

Given a chromosome image A with angles A_i and sides a_i, a quantitative measure of the similarity of this chromosome image to median chromosome images may be defined as

$$\mu_M(A) = \mu_s(A) \cdot \left[1 - \frac{|a_1 - a_4| + |a_2 - a_3|}{a_1 + a_2 + a_3 + a_4 + a_5}\right]$$

or

$$\mu'_M(A) = 1 - \mu_D(A) \cdot \frac{|a_1 - a_4| + |a_2 - a_3|}{a_1 + a_2 + a_3 + a_4 + a_5},$$

where

$$\mu_D(A) = 1 - \mu_s(A).$$

Lemma 7. Given a chromosome image A with angles A_i and sides a_j, the following three conditions are equivalent:

1. $\mu_M(A) = 1$.
2. $\mu'_M(A) = 1$.
3. Chromosome image A is a symmetrical chromosome image and $a_1 = a_2 = a_3 = a_4$.

Lemma 8. Given a chromosome image A,

$$\mu_M(A) \leq \mu'_M(A).$$

Lemma 9. Given a chromosome image A, let

$$\mu_{DM}(A) = 1 - \frac{|a_1 - a_4| + |a_2 - a_3|}{a_1 + a_2 + a_3 + a_4 + a_5}.$$

Then,

$$0 \leq \mu_M(A) \leq \mu'_M(A) \leq \mu_s(A) \leq 1$$

and

$$0 \leq \mu_M(A) \leq \mu'_M(A) \leq \mu_{DM}(A) \leq 1.$$

Lemma 10. Given a chromosome image A, let

$$L_M = \{0, \mu_M(A), \mu'_M(A), \mu_s(A), \mu_{DM}(A), 1\};$$

then the set L_M with max and min as the two binary operations forms a distributive but not complemented lattice.

E.5 Submedian Images

By using the terminal set V_T, string representation of a submedian chromosome image is as follows:

$$\chi_{(\text{submedian})} = cbabbbdbbbbbabbbcbbbabbbbbdbbbab.$$

Given a chromosome image A with angles A_i and sides a_i, let

$$a_{SM} = \min\{\,|a_1 - 2a_4| + |a_2 - 2a_3|, |2a_1 - a_4| + |2a_2 - a_3|\,\}$$

and

$$a_T = a_1 + a_2 + a_3 + a_4 + a_5.$$

A quantitative measure of the similarity of this chromosome image A to submedian chromosome images may be defined as:

$$\mu_{SM}(A) = \mu_s(A)[1 - a_{SM}/2a_T]$$

or

$$\mu'_{SM}(A) = 1 - \mu_D(A)\ (a_{SM}/2a_T).$$

Lemma 11. Given a chromosome image A, the following three conditions are equivalent:

1. $\mu_{SM}(A) = 1$.
2. $\mu'_{SM}(A) = 1$.
3. Chromosome image A is a symmetrical chromosome image and (either $a_1 = 2a_4$, $a_2 = 2a_3$ or $a_4 = 2a_1$, $a_3 = 2a_2$).

Lemma 12. Given a chromosome image A,

$$\mu_{SM}(A) \leq \mu'_{SM}(A).$$

Lemma 13. Given a chromosome image A, let

$$\mu_{DSM}(A) = 1 - a_{SM}/2a_T;$$

then

$$0 \leq \mu_{SM}(A) \leq \mu'_{SM}(A) \leq \mu_s(A) \leq 1$$

and

$$0 \leq \mu_{SM}(A) \leq \mu'_{SM}(A) \leq \mu_{DSM}(A) \leq 1.$$

Lemma 14. Given a chromosome image A, let

$$L_{SM} = \{0, \mu_{SM}(A), \mu'_{SM}(A), \mu_s(A), \mu_{DSM}(A), 1\}.$$

This set L_{SM} with max and min as the two binary operations forms a distributive but not complemented lattice.

Depending on its prospective application, the constant 2 used in defining a_{SM} may be changed to other constants.

E.6 Acrocentric Images

Given the terminal set V_T, string representation of an acrocentric chromosome image is as follows:

$$\chi_{(\text{acrocentric})} = cadbbbbbbabbbbbcbbbbbabbbbbbda.$$

Given a chromosome image A with angles A_i and sides a_i, let

$$a_{AC} = \min\{|a_1 - 4a_4| + |a_2 - 4a_3|, |4a_1 - a_4| + |4a_2 - a_3|\};$$

then a quantitative measure of the similarity of this chromosome image A to acrocentric chromosome images may be defined as

$$\mu_{AC}(A) = \mu_s(A) \cdot [1 - a_{AC}/4a_T]$$

or

$$\mu'_{AC}(A) = 1 - \mu_D(A) \cdot (a_{AC}/4a_T).$$

Lemma 15. Given a chromosome image A, the following three conditions are equivalent:

1. $\mu_{AC}(A) = 1$.
2. $\mu'_{AC}(A) = 1$.
3. Chromosome image A is a symmetrical chromosome image and (either $a_1 = 4a_4$, $a_2 = 4a_3$ or $a_4 = 4a_1$, $a_3 = 4a_2$).

Lemma 16. Given a chromosome image A,

$$\mu_{AC}(A) \leq \mu'_{AC}(A).$$

Lemma 17. Given a chromosome image A, let

$$\mu_{DAC}(A) = 1 - a_{AC}/4a_T.$$

Then

$$0 \leq \mu_{AC}(A) \leq \mu'_{AC}(A) \leq \mu_s(A) \leq 1$$

and

$$0 \leq \mu_{AC}(A) \leq \mu'_{AC}(A) \leq \mu_{DAC}(A) \leq 1.$$

Lemma 18. Given a chromosome image A, let

$$L_{AC} = \{0,\ \mu_{AC}(A),\ \mu'_{AC}(A),\ \mu_s(A),\ \mu_{DAC}(A),\ 1\};$$

then the set L_{AC} with max and min as the two binary operations forms a distributive but not complemented lattice.

Depending on its prospective application, the constant 4 used in defining a_{AC} may be changed to other constants.

E.7 An Algorithm

Algorithms for classifying a triangle as an "approximate isosceles triangle," "approximate equilateral triangle," "approximate right triangle," "approximate isosceles right triangle," or "ordinary triangle," and algorithms used to classify a quadrangle as "approximate square," "approximate rectangle," "approximate rhombus," "approximate parallelogram," "approximate trapezoid," or "ordinary quadrangle" have been presented elsewhere.[18] This section presents an algorithm for classifying a chromosome image into one of the following three classes: "approximate median," "approximate submedian," or "approximate acrocentric." This can be done in the following manner. Compute μ_M, μ_{SM}, and μ_{AC}, and set a threshold δ, where δ is a parameter and $0 \leq \delta < 1$. If we compare the max$\{\mu_M, \mu_{SM}, \mu_{AC}\}$ with δ, there are two possibilities:

1. If max$\{\mu_M, \mu_{SM}, \mu_{AC}\} > \delta$, then there are two possibilities:
   a. If the maximum is unique, then we choose the class corresponding to the maximum value and classify the image accordingly.
   b. If the maximum is not unique, then we define a priority among μ_M, μ_{SM}, and μ_{AC} and classify the chromosome image accordingly.
2. Otherwise, the chromosome image is rejected as not belonging to any of the classes.

Depending on its prospective application, μ_M, μ_{SM}, and μ_{AC} may be replaced by μ'_M, μ'_{SM}, μ'_{AC}, respectively.

E.8 Expansion or Contraction Constant σ

The expansion or contraction of a chromosome image will not affect the values of the 8 angles. Let σ be an expansion or contraction constant, with $\sigma > 0$. By applying an expansion or contraction constant σ to a chromosome image A with angles A_i and sides a_j, we obtain a new chromosome image denoted as σA, with angles A_i and sides σa_j.

Lemma 19. μ_s is invariant with respect to an expansion or contraction constant σ:

$$\mu_s(\sigma A) = \mu_s(A).$$

Definition 5. A quantitative measure of a chromosome image is a *shape-oriented* measure if and only if this measure is invariant with respect to an expansion or contraction constant σ.

Example 9. μ_s is a shape-oriented measure.

Lemma 20. μ_M, μ'_M, μ_{DM}, μ_{SM}, μ'_{SM}, μ_{DSM}, μ_{AC}, μ'_{AC} and μ_{DAC} are shape-oriented measures, and are independent of the size of chromosome images.

Lemma 21. The classification algorithm described in Section E.7 is independent of the size of chromosome images, and is a shape-oriented classification algorithm.

E.9 Conclusion of Section E

A shape-oriented classification algorithm has been described. As we have stated elsewhere,[23] there are three advantages of shape-oriented similarity measures. These are

1. Two chromosome images may have the same shape but differ in area and dimensions and still be similar.
2. Shape-oriented similarity measures can be normalized between 0 and 1.
3. Shape-oriented similarity measures are invariant with respect to rotation, translation, or expansion or contraction in size.

As Rock demonstrated,[28] the perception of form embodies the automatic assignment of a top, a bottom, and sides. Thus, orientation plays an important role in unassisted chromosome classification. Owing to the invariance of angle and length measurements with respect to orientation, shape-oriented chromosome classification would not be confused by the orientation of the chromosome. In this sense, shape-oriented chromosome classification is better than unassisted chromosome classification. The results obtained in this section may have useful applications in the storage, retrieval, and classification of chromosome images and geometric figures. In addition, the results may be applied to other areas, for instance, to the work done by Pavlidis[29–31] in the area of approximating an arbitrary shape by polygons, the work done by Dacey[32] in the development of a two-dimensional language that produces line pictures of polygons, or the work presented by Harmon[33] on face recognition. The results may also be of use in pattern recognition, information retrieval and artificial intelligence.

We conclude this section with another proposition. (As shown in Figure 3(d), (A_1, A_2) and (A_3, A_4) are exterior biangles; (A_5, A_6) and (A_7, A_8) are interior biangles.)

Proposition 5. For shape-oriented storage of chromosome images, if we logically order all the chromosome images individually and independently according to the angular sums of exterior biangles and interior biangles, then we can reduce retrieval time for answering queries such as "retrieve the chromosomes that are very very similar to a given chromosome A."

F. DECISION SUPPORT SYSTEMS INVOLVING LEUKOCYTE IMAGES

Let the universe of discourse $U_L = \{L\}$ be the universe of leukocyte images. Let the set of terms T_L be leukocytes with circular nuclear shape, leukocytes with elongated nuclear shape, leukocytes with spiculed nuclear shape, leukocytes with indented nuclear shape, leukocytes with slightly indented nuclear shape, and leukocytes with deeply indented nuclear shape.

The peripheral blood leukocytes have been classified into eight categories.[34] The categories are small lymphocytes, medium lymphocytes, large lymphocytes, band neutrophils, segmented neutrophils, eosinophils, basophils, and monocytes. Pictures of typical peripheral blood leukocytes are given by Bacus and Gose.[34] The features they used were nuclear size, nuclear shape, nuclear and cytoplasmic texture, cytoplasm color, and cytoplasm colored texture.

As Bacus and Gose stated, it is particularly important not only to classify the five major cell types, but also to determine *intraclass* differences, between younger and older cells in some classes.[34] For example, intraclass percentage shifts relate to the production rates and maturation of new cells, and thus to the physiological response to stress. In this section, our attention will be focused on the study of the shape properties of leukocytes in order to determine intraclass differences.

Young has stated that the nuclear shapes of lymphocytes, monocytes, eosinophils and basophils are round, indented, segmented, two-lobed and elongated, respectively.[35] Four shape features (circular, spiculed, oblong and irregular) were listed by Bacus and Aggarwal as typical features for computerized microscopic-image analysis; however, they did not show how to measure these four shape features.[36]

The nuclear-shape measure used by Bacus and Gose was (perimeter)^2/area.[34] Let f denote this measure. Then,

$$f = (\text{perimeter})^2/\text{area}.$$

Bacus and Gose give a way to obtain perimeter and area measure.[34]

Out of all possible nuclear shapes, circular shapes minimize f; the minimum of f is equal to 4π.

F.1 Equal-Perimeter Circular-Shape Measure

The range of f is from 4π to infinite. In order to normalize this measure between 0 and 1, a normalization constant ρ_1 is multiplied times the reciprocal of f as:

$$\mu_1 = \rho_1/f$$

where ρ_1 is a normalization constant, and is set to 4π. μ_1 may be viewed as a quantitative measure of the meaning of an "approximately circular shape" or the meaning of an "approximately round shape."

F.2 Equal-Area Circular-Shape Measure

Let P and A denote, respectively, the perimeter and the area of a nucleus. Let P' denote the perimeter of a circle with area A. Let A' denote the area of a circle with perimeter P. Depending on its prospective application, the meaning of an "approximately circular shape" may also be expressed as:

$$\mu_2 = \rho_2 P'/P,$$

where ρ_2 is a normalization constant, and is set to 1 in order to normalize the value of μ_2 between 0 and 1.

μ_1 may be called an equal-perimeter circular-shape measure and μ_2 an equal-area circular-shape measure. The relationship between μ_1 and μ_2 is summarized in the following three theorems.

Theorem 3. For all possible nuclear shapes, the equal-perimeter shape measure μ_1 is always less than or equal to the equal-area circular-shape measure μ_2.

Theorem 4. For all possible nuclear shapes, the equal-perimeter circular-shape measure μ_1 is always equal to the square of the equal-area circular-shape measure μ_2.

Theorem 5. For any two nuclear shapes A and B, $\mu_1(A)$ is greater than or equal to $\mu_1(B)$ if and only if $\mu_2(A)$ is greater than or equal to $\mu_2(B)$.

The proofs of theorems can be obtained from the author. Theorem 5 shows the order-preserving property of μ_1 and μ_2 with respect to all possible nuclear shapes.

F.3 Elongated-Shape Measures

The nuclear shape of the basophil cell type is elongated. A quantitative measure of the elongated visual concept of a nucleus may be defined as follows:

1. Determine its area and center of mass. A way to determine its area and center of mass is given by Bacus and Gose.[34]

2. Find the best-fit rectangle with the same area and with its center coinciding with the center of mass of the nucleus.
3. Let a and b denote the sides of the best-fit rectangle. Let μ_e' denote a quantitative measure of the elongated visual concept of the nucleus. Then

$$\mu_e' = \frac{\max(a, b)}{a+b}.$$

4. If the best-fit rectangle is not unique, the μ_e is defined to be the maximum of μ_e'—

$$\mu_e = \max(\mu_e').$$

F.4 Spiculed-Shape Measures

A quantitative measure of the spiculed visual concept of a nucleus may be expressed as

$$\mu_{sp} = \mu_e^2.$$

Depending on its prospective application, the exponent used in defining μ_{SP} may take on different values.

F.5 Indented-Shape Measures

The nuclear shape of the monocyte cell type is indented. A quantitative measure of the visual concept "indented," as used of a nucleus, may be defined as follows:

1. As shown in Figure 4, determine points A and B such that the indented nucleus is symmetric with respect to AB. If the symmetric axis AB does not exist, determine the axis AB that will minimize the symmetrical difference.
2. Determine points C and D such that the tangents at C and D are perpendicular to the tangent at A. If C is not unique, then determine the middle point as C.
3. Determine points E and F such that AE is equal to EC, and AF is equal to FD along the perimeter.
4. Determine θ_1, which is the angle formed by the tangents at points E and F. θ_1 may be called the *exterior angle* of an indented nucleus.
5. Determine points G and H such that the tangents at G and H are parallel to the tangent at A. If G is not unique, then determine the middle point as G.
6. Determine points I and J such that BI is equal to IG, and BJ is equal to JH along the perimeter.
7. Determine θ_2, which is the angle formed by the tangents at points I and J. θ_2 may be called the *interior angle* of an indented nucleus.
8. Let μ_i denote a quantitative measure of the indented visual concept of the nucleus. Then,

$$\mu_i = 1 - \rho_i \max(\theta_1, \theta_2)$$

or

$$\mu_i' = 1 - \rho_i' \min(\theta_1, \theta_2)$$

or

$$\mu_i'' = 1 - \rho_i'' \theta_1 \theta_2,$$

where ρ_i, ρ_i', and ρ_i'' are normalization constants, set to 1/180°, 1/180° and 1/(180°)^2, respectively, in order to normalize the values of μ_i, μ_i', and μ_i'' between 0 and 1.

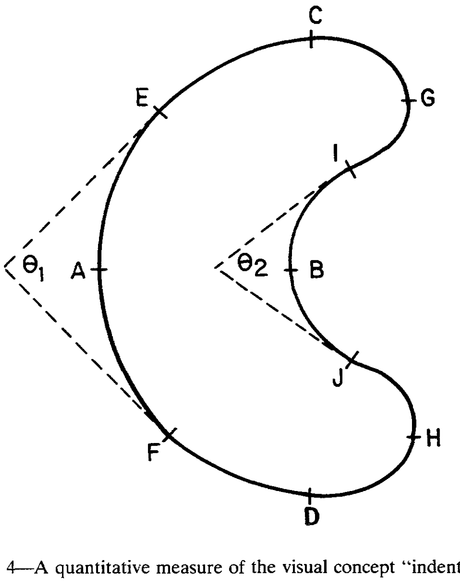


Figure 4—A quantitative measure of the visual concept "indented"

Example 10

1. For Figure 5(a), $\theta_1 = \theta_2 = 0°$. Thus, $\mu_i = 1$.
2. For Figure 5(b), $\theta_1 = 25°$ and $\theta_2 = 10°$. Thus, $\mu_i = 0.86$.
3. For Figure 5(c), $\theta_1 = 84°$ and $\theta_2 = 70°$. Thus, $\mu_i = 0.53$.
4. For Figure 5(d), $\theta_1 = 180°$ and $\theta_2 = 180°$. Thus, $\mu_i = 0$.
5. For Figure 5(e), $\theta_1 = 90°$ and $\theta_2 = 180°$. Thus, $\mu_i = 0$.

It is of interest to note that μ_i is monotone decreasing with respect to the exterior angle θ_1 and the interior angle θ_2.

F.6 Slightly Indented Shape Measures

A quantitative measure of the slightly indented visual concept of a nucleus may be expressed as

$$\mu_{si} = \mu_i^{1/2}.$$

Example 11. (1) For Figure 5(b), $\mu_{si} = 0.93$. (2) For Figure 5(c), $\mu_{si} = 0.73$.

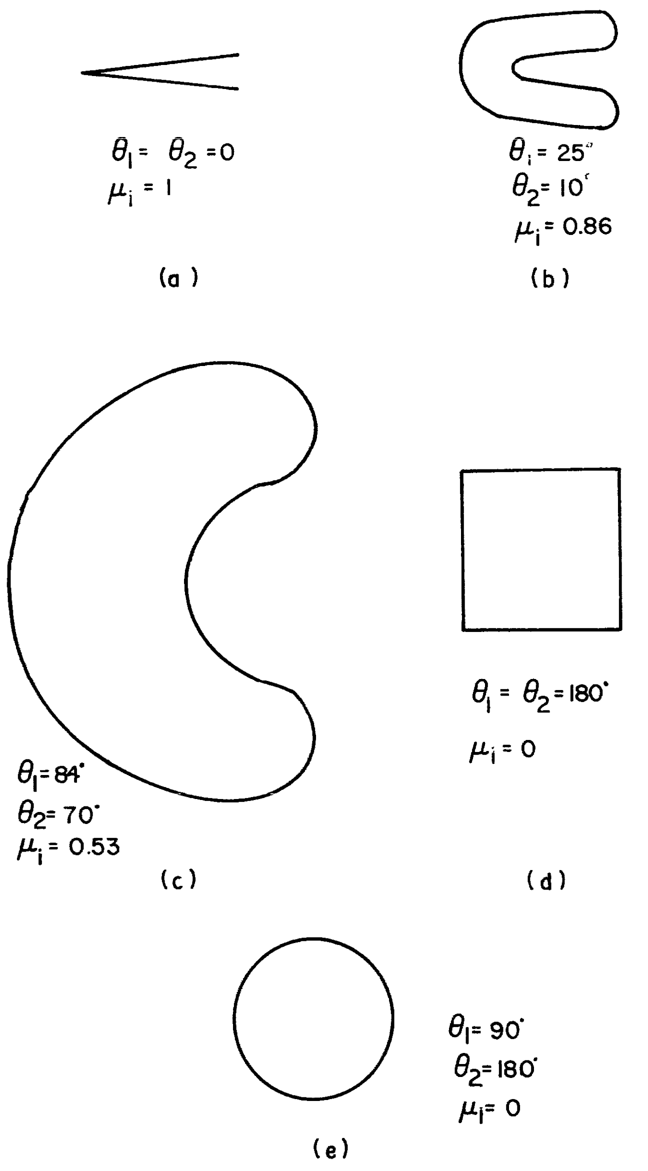


Figure 5—The grade membership of indented nuclei

Depending on its prospective application, the exponent used in defining μ_{si} may take on different values.

F.7 Deeply Indented Shape Measures

A quantitative measure of the visual concept "deeply indented," for a nucleus, may be expressed as

$$\mu_{di} = \mu_i^2 .$$

Example 12. (1) For Figure 5(b), $\mu_{di} = 0.74$. (2) For Figure 5(c), $\mu_{di} = 0.28$.

Depending on its prospective application, the exponent used in defining μ_{di} may take on different values.

Proposition 6. For shape-oriented storage of leukocytes, it is suggested that leukocytes be logically ordered individually and independently according to the magnitudes of the grade of membership of approximately circular (round) nuclei, elongated nuclei, indented nuclei, slightly indented nuclei, and deeply indented nuclei, so that, for answering queries, the amount of leukocyte data that must be searched can be reduced and the response time improved.

G. CONCLUSIONS

The foregoing analysis has shown that the concepts of fuzzy languages and pictorial databases[37] can be applied to decision support system design methodologies. Decision support systems involving geometric figures, chromosome images, or leukocyte images are presented as illustrative examples.

In similarity retrieval from a pictorial database, very often it is desired to find pictures (or feature vectors, histograms, etc.) that are most similar to or most dissimilar to a test picture (or feature vector). Algorithms for finding most dissimilar images may be found elsewhere.[38] The unique pair of the most dissimilar chromosome images is shown in Figure 6.

Earlier, a review and evaluation of the state of the art of similarity retrieval and updating techniques was presented[13]; various types of similarity measures for extracting picture primitives were described. Using similarity measures, one can not only store similar pictures logically or physically close to each other to improve retrieval efficiency, one can also use such similarity measures to answer fuzzy queries involving nonexact retrieval conditions.

The applications of fuzzy languages and pictorial databases to decision support system design methodologies offer what appears to be a fertile field for further study. The underlying ideas are interesting and easy to apply practically. The results have useful applications in decision support systems, pattern recognition,[39–46] pictorial information systems and artificial intelligence.

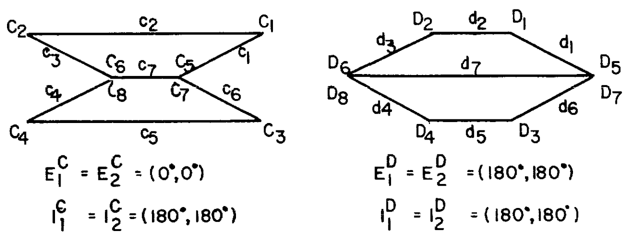


Figure 6—The unique pair of the most dissimilar chromosome images

REFERENCES

1. Lee, D. T. "Database Design for Decision Support Systems." In S. K. Chang (Ed.) *Management and Office Information Systems.* New York: Plenum, 1983, in process.
2. Lee, D. T., et al. "Quintary Tree: A File Structure for Multidimensional Database Systems." *ACM Transactions on Database Systems,* 5 (1980).
3. Lee, D. T. "The Contingent Model of Decision Support Systems." *The Proceedings of Western AIDS,* 1982.

4. Lee, D. T. "The Unified Approach for Designing Decision Support Systems." *DSS-82 Transactions,* 1982.
5. Lee, D. T. "United Database for Decision Support." *International Journal of Policy Analysis and Information Systems,* (1982).
6. Mackenzie, K. D. "Organizational Structures as the Primal Information System: An Interpretation." In S. K. Chang (Ed.) *Management and Office Information Systems.* New York: Plenum, 1983, in process.
7. Blanning, R. W. "Language Design for Relational Model Management." In S. K. Chang (Ed.), *Management and Office Information Systems.* New York: Plenum, 1983, in process.
8. Pan, S., R. A. Pick, and A. Whinston. "Database Management Support for Decision Support Systems." In S. K. Chang (Ed.), *Management and Office Information Systems.* New York: Plenum, 1983, in process.
9. Ellis, C. A., and G. J. Nutt. "Computer Science and Office Information Systems," Technical Report SSL-96-6, Xerox Palo Alto Research Center, December 1979.
10. Date, C. J. *An Introduction to Database Systems* (3rd ed.). Reading, Mass.: Addison-Wesley, 1981.
11. Codd, E. F. "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM,* 14 (1970).
12. Lee, E. T., and L. A. Zadeh. "Note on Fuzzy Languages." *Information Sciences,* 1 (1969), pp. 421–434.
13. Lee, E. T. "Similarity Retrieval Techniques." In S. K. Chang and K. S. Fu (Eds.), *Pictorial Information Systems.* New York: Springer-Verlag, 1980, pp. 128–176.
14. Hopcroft, J. E., and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Reading, Mass.: Addison-Wesley, 1979.
15. Rosenkrantz, D. J. "Matrix Equations and Normal Forms for Context-Free Grammar." *Journal of the ACM,* 14 (1967), pp. 501–507.
16. Bellman, R. E., and L. A. Zadeh. "Decision-Making in a Fuzzy Environment." *Management Science,* 17 (1970), pp. B-141–B-164.
17. Lee, E. T. "Algorithms for Finding Chomsky and Greibach Normal Forms for a Fuzzy Context-free Grammar Using an Algebraic Approach." *International Journal of Cybernetics and Systems,* 1983, in process.
18. Lee, E. T. "Proximity Measures for the Classification of Geometric Figures." *Journal of Cybernetics,* 2 (1972), pp. 43–59.
19. Zadeh, L. A. "Similarity Relations and Fuzzy Ordering." *Information Science,* 3 (1971), 159–176.
20. Zadeh, L. A. "A Fuzzy-Set-Theoretic Interpretation of Linguistic Hedges." *Journal of Cybernetics,* 2 (1972), 4–34.
21. Chu, Y. *Computer Organization and Microprogramming.* Englewood Cliffs, N.J.: Prentice-Hall, 1972.
22. Richards, H., and A. E. Oldehoeft. "Hardware-Software Interactions in Symbol-2R's Operating System." *Second Annual Symposium on Computer Architecture,* 1975.
23. Lee, E. T. "The Shape-Oriented Dissimilarity of Polygons and Its Application to the Classification of Chromosome Images." *Pattern Recognition,* 6 (1974), pp. 47–60.
24. Stohr, E. A., and N. H. White. "Software for Planning and Decision Support Systems: An Overview." In S. K. Chang (Ed.), *Management and Office Information Systems.* New York: Plenum, 1982.
25. Wildrow, B. "The Rubber-Mask Technique—I: Pattern Measurement and Analysis." *Pattern Recognition,* 5 (1973), pp. 175–197.
26. Lee, H. C., and K. S. Fu. "A Stochastic Syntax Analysis Procedure and Its Application to Pattern Classification." *IEEE Transactions on Computers,* C-21 (1972), pp. 660–666.
27. Ledley, R. S., L. S. Rotolo, T. J. Golab, J. D. Jacobsen, M. D. Ginsberg, and J. B. Wilson. "FIDAC: Film Input to Digital Automatic Computer and Associated Syntax-Directed Pattern-Recognition Programming System." In *Optical and Electro-Optical Information Processing.* Cambridge, Mass.: MIT Press, 1965, pp. 591–613.
28. Rock, I. "The Perception of Disoriented Figures." *The Scientific American* (1974), pp. 78–85.
29. Pavlidis, T. "Analysis of Set Patterns." *Pattern Recognition,* 1 (1968), 165–178.
30. Pavlidis, T. "Computer Recognition of Figures Through Decomposition." *Information and Control,* 12 (1968), pp. 626–637.
31. Pavlidis, T. "Representation of Figures by Labeled Graphs." *Pattern Recognition,* 4 (1972), pp. 5–17.
32. Dacey, M. F. "Poly: A Two Dimensional Language for a Class of Polygons." *Pattern Recognition,* 3 (1971), pp. 197–208.
33. Harmon, L. D. "The Recognition of Faces." *The Scientific American,* 229 (Nov. 1973), pp. 71–82.
34. Bacus, J., and E. Gose. "Leukocyte Pattern Recognition." *IEEE Transactions on Systems, Man and Cybernetics,* SMC-2 (1972), pp. 513–526.
35. Young, I. "Automated Leucocyte Recognition." In *Automated Cell Identification and Cell Sorting.* New York: Academic Press, 1972.
36. Bacus, J., and Aggarwal, et al. "Computer Recognition of Microscopic Images." *Proceedings of EASCON,* 1975.
37. Lee, E. T. "Similarity Retrieval for Pictorial Databases." In S. K. Chang (Ed.), *Management and Office Information Systems.* New York: Plenum, 1983, in process.
38. Lee, E. T. "Algorithms for Finding Most Dissimilar Images with Possible Applications to Chromosome Classification." *Bulletin of Mathematical Biology,* 38 (1976), pp. 505–516.
39. Lee, E. T. "An Application of Fuzzy Sets to the Classification of Geometric Figures and Chromosome Images." *Information Sciences,* 10 (1976), pp. 95–114.
40. Lee, E. T. "Shape-Oriented Chromosome Classification." *IEEE Transactions on Systems, Man, and Cybernetics,* SMC-5 (1975), pp. 629–632.
41. Lee, E. T. "Shape-Oriented Classification Storage and Retrieval of Leukocytes." *Proceedings of the Third International Joint Conference on Pattern Recognition,* Nov. 8–11, 1976.
42. Lee, E. T. "Shape-Oriented Storage and Retrieval of Geometric Figures and Chromosome Images." *Information Processing and Management,* 12 (1976), pp. 35–41.
43. Lee, E. T., and N. Lee. "Design of a Fault Tolerant Microprocessor." *Proceedings of the 1978 International Computer Symposium,* Dec. 20–24, 1978, pp. 100–101.
44. Lee, E. T. "A Shape-Oriented Image Data Base." *Proceedings of the Symposium on Current Problems in Image Science,* Monterey, Calif.: Nov. 1976.
45. Lee, E. T. "A Similarity Directed Picture Database." *Policy Analysis and Information Systems,* 1978, pp. 113–125.
46. Lee, E. T. "Similarity-Directed Picture Storage and Management." *Proceedings of the 1977 IEEE Workshop on Picture Data Description and Management,* 1977.

Database-oriented decision support systems

by DANIEL T. LEE
University of Hartford
West Hartford, Connecticut

ABSTRACT

Decision support systems represent a new challenge for computer technology experts, management science specialists, and organizational theorists. This paper examines the related topics on the database design for decision support systems. The major emphasis is on the global conceptual model and the unified approach for database design. The entity-relationship model is used for illustrating the unified approach, and the various translation processes are also discussed. A practical example is given. Throughout the paper, the interfaces among the user, the model, and the database are stressed.

INTRODUCTION

The computer has been a source of expectation since the first commercial electronic computer, the UNIVAC I, became available in 1951. During the past three decades, though the computer has made great contributions toward civilization, it has been limited basically to transaction processing, focusing on data storage and flow. Decision makers have been convinced that the computer can enhance their organization's efficiency by reducing cost and increasing profit, but are not convinced that it can do much to improve their decision making. It is only recently that the computer has been considered an integral part of the decision-making process.[27]

The computer has evolved through three stages. Initially, the computer was employed in routine data processing (e.g., billing, payroll, and scheduling) and producing reports. It was soon determined that the various functions played by this electronic data processing (EDP) stage should be integrated for the sake of efficiency. Unfortunately, the use of EDP for executives' decision making has been indirect, as Keen and Morton[26] point out, because its major use was providing reports and access to data. EDP failed to meet the expectations of the management information system (MIS), according to Wagner.[49] However, since the payoff of improving executives' decision-making is high, both academicians and practitioners are trying to change the MIS to improve the decision-making process. Decision support systems (DSS) are incorporated with data management capability and model handling capability in the decision-making process.[43, 8, 24]

De and Sen[18] are strongly against the decision-specific approach exemplified in the studies of Keen and Morton[26] and Alter.[3] De and Sen believe that such approaches suffer from a common problem: they are not general. Because it is difficult to predict the types of decisions executives will make and there are unpredictable changes in the environment, it would be inappropriate to say that the DSS should be data-oriented or model-oriented.[3] Though executives need information that is largely unstructured and externally oriented, they also need structured and internal reports. According to Haseman,[24] the types of information required for decision makers can be classified as (1) repetitive report generation, (2) ad hoc language capability, and (3) model interface capability. Bonczek and coauthors[8] also indicated that a decision support system should address at least the three topics of data management—modeling (computation management), the user's interface (i.e., the user's language), and data management. These three are intertwined and interdependent. In general, a DSS is an integrated set of data and models. It should be designed with a unified database. It should be easy to use, flexible enough to meet all types of information requirements at all levels, and dynamic enough to meet all changes in the future. To meet these above requirements, the role of an integrated database is crucial. Unfortunately, such an integrated database has not been designed. This paper tries to close this gap by designing a unified database that will meet the decision maker's needs in satisfying data management and model-handling capabilities.

THE DESIGN CRITERIA FOR A DECISION SUPPORT SYSTEM

Morton[38] first characterized a DSS as an interactive computer system incorporated with data and models for solving mainly unstructured problems for decision makers. Bonczek, Holsapple, and Whinston[8] outlined five design criteria as proposed by Sprague and Watson.[41] They are:

1. A set of models supports decision making in various functional areas and at various management levels.
2. Models are devised as modules that can stand alone or be used in conjunction with one another to form more models.
3. A query language can interact with the database.
4. A mechanism for models extracts data from a database.
5. Flexibility in modifying the modules meets the requirement of a changing environment.

These criteria exemplify a DSS having the capability of interfacing databases and models interactively. Actually, the major function of a DSS is to support the decision-making process. Decision makers also need information from standard reports that might be produced in batch processing or online processing. A genuine DSS should be a system incorporated with data management capability and modeling capability in batch or interactive processing.

DATABASE DESIGN

Design Constraints

The current technology in database design exists generally for static and stable environments. There are very few, if any, design methodologies that are readily available for designing such a comprehensive and versatile database for executives. This paper examines such a design methodology in the next few sections.

Conceptualization of the Organization

Information is increasingly becoming a scarce resource for an organization. Efficient management and use of this scarce resource determines successful operation of an organization. For efficient use of information, the information requirements for various decisions should be identified through a conceptual model that is used as a communication tool between the system designer and the user. This conceptual model can be divided into local conceptual models and a global conceptual model. The local conceptual model is developed by examining the information requirements for the various decisions (or talking to the personnel or the decision maker concerned). Then all these local conceptual models are integrated into a global conceptual model. The global model is used without consideration of the physical aspects of the database. The local conceptual views are equivalent to the external schema in the ANSI-X3-SPARC/DBMA[4] Interim Report, which consists of three distinct types of schemas: external, conceptual, and internal. The external schemas are the definitions of the user's views of a database and their mapping to the conceptual schema (or global conceptual view of the organization). The conceptual schema is the definition of the real world (or the organization's view, or DBA's view) of the database and its mapping to the internal schemas. The internal schema is concerned with the physical storage and access path.[34,45]

The major reasons for using conceptual schema as the interface[11] are:

1. It reduces the number of mappings between the external schemas and internal schemas.
2. Since the conceptual schema is not concerned with the physical storage and access path, it is relatively stable while allowing changes in external schemas and internal schemas, and it plays an important role in maintaining logical data independence and physical data independence.
3. The conceptual schema is the same as the enterprise schema.

The proposed entity-relationship diagram[10] can be used to describe the conceptual schema. Furthermore, the external schemas may be expressed in terms of three logical data models—relational, hierarchical, and network (DBIG). The E/R diagram will be a very useful tool for translating the conceptual model of data into various logical data models. This paper follows the basic concepts and the translation rules of the Entity-Relationship Model (E/R model) in defining its conceptual model and subsequent mapping between the conceptual model and logical data models, because the E/R model is very mature in defining the conceptual schema of an organization, and its mechanism is easy to understand.

Logical Data Models

The logical view of data has been an important issue in recent years in database design, because existing commercial systems are based on one of the three major data models (relational, hierarchical, and network)[15] and almost exclusively based on either hierarchical or network models.[47] This situation might change soon toward a relational approach or a unified approach. These subjects are discussed in the next few sections. When designing a database, problems of both computer efficiency and human efficiency should be considered.[48] In terms of human efficiency, data should be presented to a user in a form suited to both the user's skills and the application required. The choice of data structure to be supported at the user level (external or conceptual) critically affects many components of the system. It also dictates the design of the corresponding data manipulation language(s), because each DML operation must be defined in terms of its effect on those data structures.[17] Thus the question "Which data structures and associated operators should the system support?" is a crucial one. Before answering this question, the three best-known approaches (relational, hierarchical, and network) will be examined.

The Relational Data Model

The relational model is based on relational theory. The mathematical concept underlying the relational model is the set-theoretic relation, which is a subset of the cartesian product of a list of domains. A domain is a set of values. The relational model consists of a collection of relations, each of which can be thought of as a simple table. Rows of such tables are generally referred to as tuples, which correspond to entities. Columns are usually referred to as attributes. The ordering of rows and columns is immaterial. The attribute name usually corresponds to domain name, provided no ambiguities occur; otherwise, a role name should be used to qualify the domain name as the attribute name. The domain and the attribute are different concepts. The former is a set of values; the latter is a list of items taken from the domain. A particular item can occur several times in the column of attribute values, while it occurs only once in the set of values in a domain. Sometimes, two or more attributes take values from the same domain. The attribute in a relational model is used to distinguish domains with the same name in the same relation and to map attribute names to values in the domains of the attributes.

The list of attribute names for a relation is called the relation scheme, which corresponds to a record format. A relation corresponds to a file and a tuple to a record. The collection of relations with their values is called the relational database. We are free to create relations with any set of attributes from the relation schemes. Since the number of basic constructs in the relational data model is one (namely, the relation itself), all information in the database is represented by this simple construct. We need only one operator for each of the four basic functions—retrieve, insert, delete, and update. In order to avoid the operation anomalies, the relational model is usually operated under normalized relations. It possesses a body of theory applicable to database problems, such as normalization theory for designing a relational schema,[15,51] and the theory of relational completeness for comparing relational languages.[17]

In summary, the structure of a relational model is best by describing each domain, with its possible data type, and each relation, identifying its attributes and primary key, which uniquely identifies each type within a relation. Tuples correspond to entities, and relations correspond to entity set. Both the entities and relationships among entities are represented by one construct, relation, one-to-many and many-to-many relationships are treated the same.

It is clear that relational structure is simple and easy to understand, and it is predicted that it will become the major contender in the logical data model selection.

The Network Data Model

The specification of the CODASYL Data Base Task Group (DBTG) for defining and processing network-oriented databases was recorded in the 1971 Report of CODASYL DBTG,[12] and the discussion of the network data model is detailed by Olle.[40] This report,[12] written by a group of voluntary representatives from computer manufacturers, users in U.S. industry and government, and university researchers has become the basis for some sort of standardization of a generalized database management system.

The basic constructs of a network data model are logical record types represented by rectangles that correspond to entity sets and DBTG-set types represented by arrows that correspond to relationship types in the E/R model (which are discussed in detail in the following sections). The most important construct in the network model is the set that goes from the owner record type to one or more member record types. In case of many-many relationships, it must create a link record type between the two record types that are the owner record types of the link record type, and then these two record types can be one-one correspondence. According to DBTG Data Definition Language (DDL), one-many relationships are usually required; of course, it can be one-0, one-one, or one-many. Record occurrences correspond to entities, data items to attributes, and set occurrences to relationships.

The network model compared with the relational model shows that repeating groups are allowed in the network model, while they are not allowed in the latter; the former does not have to have a tuple identifier, while the latter must have it; the former uses predefined access paths through set mechanisms, while the latter does not and all possible routes are dynamically materialized; the former specifies certain integrity constraints to the data structure, while the latter does not and only declares them as adjuncts to the data structure. According to Michaels,[37] four major areas of data definition, structure, physical placement, access path, and integrity and privacy constraints, the relational model includes neither physical placement nor access paths, and there is clear separation between structure and constraints, while the network model includes all these four areas of description. From these structure comparisons, the network structure is more complex than the relational. From DSS point of view, the relational model is more appealing than the network. Current implementation of database management systems, however, exists either in the network model or hierarchical model, and many inadequacies, as Michaels pointed out, are not inherent to the network approach. We might still have to deal with the network structure either by transformation between the relational and the network or by a unified approach. These are subjects of the next section.

The Hierarchical Data Model

Hierarchies are special cases of networks; any networks can be transformed into hierarchical structure. The data is represented by a tree structure. The record type at the top of the tree is known as the root or the parent, and the elements at the lowest levels, which have no children, are called leaves or children. The root may have any number of leaves; each of these may have any number of lower-level leaves. A tree shall be a hierarchy of records, because the root and the leaf are both organized as records. This has led to the term *hierarchic database*. No child record type can have more than one parent segment type, but it is allowed more than one parent (owner) in the network structure as long as the child is in a different set. A set type represents the relationship between the owner record type and member record type in a network sense, and it is formally declared; but it is implied in the hierarchical model. Many-many relationships also cannot be handled directly. It must be treated by creating a pointer segment for each of the two segments (IMS's terms of IBM) while in network structure; it is handled by creating link record types (or two-way connectors). No child segment occurrence can exist without its parent, while it is allowed in the network data model by a singular set for which the owner is the "SYSTEM," where the SYSTEM has only one occurrence in the network model.

From this comparison between a network model and a hierarchical model, we can visualize that the hierarchical structure is even more complex and rigid than the network structure. When more record types are brought into the structure, it becomes worse. The user, as Date[17] pointed out, is forced to devote more time and effort to solving problems that are introduced by the hierarchical data structure and that are not intrinsic to the questions being asked. Of course, hierarchies are a natural way to model truly hierarchical structures from the real world. Besides, virtual logical record types are very useful mechanisms in connecting two physical databases or in converting the network structure into hierarchical ones. From an updating operation point of view, however, the hierarchical model possesses even more undesirable properties than the previous two logical data models. As mentioned before, because many current implementations of databases are in the hierarchical model, we probably still have to deal with them either by transformation among the three approaches or by adopting a unified approach of logical data models.

A Comparison of the Data Models

So far, three logical data models (relational, network, and hierarchical) have been briefly reviewed in order to lay the foundation for the subsequent analysis. We hope it will give some guideline for comparisons among them to see which is the best fit for the environment of a DSS. There is a diverse community of users facing a DSS. The DSS should therefore provide information to all levels of management in a batch or in an interactive mode. The users (including decision makers at all levels, researchers, and analysts) need standard reports and interaction with the database for data retrieval, modeling, or computation in order to solve structured, semi-structured, or unstructured problems. Based on these requirements, the three logical data models are further examined.

As mentioned, the hierarchical structure is a special case of the network, and the comparison between the relational and the network approaches suffices. Michaels[37] outlines five factors for evaluation. Codd[15] and Ullman[47] each use two factors. No matter how many factors are picked up for evaluation, being easy to use, easy to understand, easy to manipulate, and easy to implement are necessary for a database designed for meeting the diverse requirements of a DSS.

From these four points of view, the relational structure meets all these requirements except "easy to implement." As Ullman[47] indicates, the multilist data structure and the pointer-based implementation of variable length records do not generalize readily to many-many mapping. Theoretically, it is always possible to implement relations by a hierarchical or network structure, but it is not always clear how to do so, especially in a cost-effective way. Otherwise, the relational model is a very promising approach because of its non-procedural languages, its separating structural elements from physical elements to maintain data independence, and its simplicity in data structure and data manipulation.

The network model is more complex than the relational because of its procedural language, its rigid structure, and its bundling four areas of data definition into one construct, which make it lack data independence. Its merit is that it is readily available for implementation. Many of the inadequacies are not inherent in the network approach. The initial DBTG specifications have undergone subsequent development and refinement as reported by two CODASYL groups.[13,14]

Generally, the relational model is more appealing in the environment of a DSS, but we are not suggesting that any one model should be dominant.[15,29,37] Because user demands are so diversified, no one model can meet the needs of users completely for all circumstances.

The Unified Approach

Many researchers have made contributions in this area.* Some examples are Date's Unified Database Language (UDL),[17] Chen's Entity-Relationship Model,[10] Zaniolo's multiple external schemas for designing and supporting relational and hierarchical views over CODASYL network schemas,[50] Vetter's structure-type coexistence,[48] and Deen's global schema for supporting interfaces to other systems.[19] There are generally two types of data structure coexistence: top-down and bottom-up. The bottom-up approach is to create normalized relations as the basic ingredients, and then derive the three data structures (relational, hierarchical, and network) from them.[17] The top-down approach[10] proceeds in an entity-relationship model as a generation from which the three existing data models can be derived. Either approach is effective. The latter approach uses the semantic information to organize the data in entity-relationship relations, which are similar to 3NF relations (as Chen[10] puts it), but with clear semantics and without using the transformation process. Besides the normalization process can get rid of only a certain degree of operation anomalies, and sometimes it becomes a messy and tedious operation in the transformation process.

The Unified Database Language (UDL) proposed by Date,[17] is very attractive and theoretically sound. It intends to support the three data structures (relational, hierarchic, and network) in a uniform way. The "onion-layer" structure is the basis for the construct of the UDL. It indicates that the language features, the operators, and the operands of a relational database are the subset of those required to declare a hierarchical database, and these in turn are a subset of those required to declare a network database.

Although the relational structure is the most promising approach, the recent development in the relational database system (RDS), such as system R, is encouraging.[9,17] It is capable of supporting both repetitive transactions and ad hoc queries that are very attractive for the needs of a DSS. The network structure and the hierarchical structure will continue in use for some time because of configuration compatibility of data processing structure in the organization. Besides, a tremendous investment has been made in these two approaches. All these lead to the development of data-structure-type coexistence. Though there is no DBMS that can support the three approaches, a strong foundation has been built up, as Date[17] pointed out; and theoretically the structure-type coexistence is feasible and desirable for the diversified community of users.

THE ENTITY-RELATIONSHIP APPROACH

The Basics of the E/R Model

As mentioned earlier, the entity-relationship model (E/R model), proposed by Chen,[10] is very close to the conceptual model (or the global view) of the organization. The entity and relationship relations in the E/R model are smiilar to 3NF relations but with clear semantics and without using the transformation operation. The steps in logical database design following the E/R approach are simple and close to human thinking. The E/R diagram is independent of the processing frequencies. Therefore, it is more stable. The translation process from an E/R diagram into logical data structures—hierarchical, network, and relational is also easy. For these reasons, the E/R model is adopted for demonstrating the logical database design concept and the transformation process from the E/R diagram into the three major logical data structures. Because the user's environment of a DSS is di-

*See References 2, 6, 10, 17, 19, 25, 29, 36, 39, 45, 46, 48, and 50.

verse, a powerful approach such as the E/R model is required and is the best fit in such an environment. In addition to its simplicity in design, its capability in synthesizing the three major logical data models into a unified approach that can meet the requirements of a DSS is the most appealing.

Procedures in Logical Database Design

It was mentioned that following the E/R model for logical database design was convenient. Let us illustrate this by an example of technological transfer. Suppose that a company has many projects. Each project needs technologies that are available through the company's own capability in a capability file or that have to be purchased from outside organizations domestically or internationally. Some technologies are not readily available in the market, and have to be sought through a potential market, so that a potential supplier list can be built up. If the necessary technologies are not available in a company's own capability file, which can be drawn out for immediate use, and developing the technologies is cheaper than purchase from the outside, the company must develop them.

For simplicity, we list only five entity record types and four relationship record types in the E/R diagram in Figure 1. The five entity record types (PROJ, TECH, SUPP, FILE, and DEVE) are represented by rectangular boxes. The four relationship record types (PROJ-TECH, SUPP-TECH, POTENT-SUPP, and TECH-AVFI) are represented by diamond boxes. These relationships can be classified according to their association types between the entity types as one-one (1:1), one-many (1:N), and many-many ($M:N$).

After identifying entity types and relationship types and drawing the E/R diagram according to their semantic meaning, the attributes and value types within each entity record type and each relationship record type should be identified as shown in Figures 2(a) and 2(b).

The next step, a data-structure diagram in Figure 3, can be derived from the E/R diagram in Figure 1. All the entity types and relationship types in the E/R diagram become record types in Figures 4 and 5 in the data-structure diagram. One-many relationships are translated into a data-structure set (i.e., an arrow), and many-many relationships need linking records that can translate the many-to-many relationships into one-to-one relationships.

The last step is to group attributes into records and decide how to implement the data-structure sets by using pointer arrays, chains, or other methods. For decision support

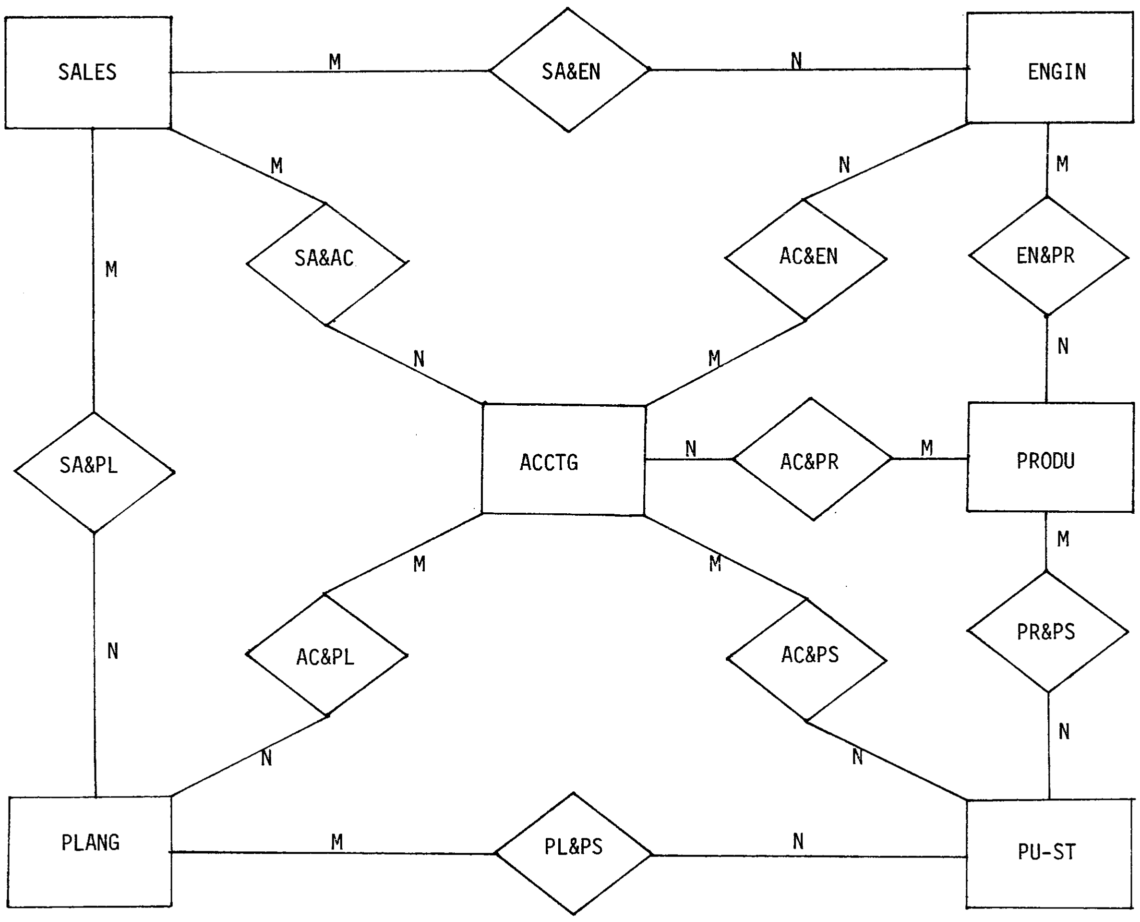


Figure 1—Entity-relationship diagram

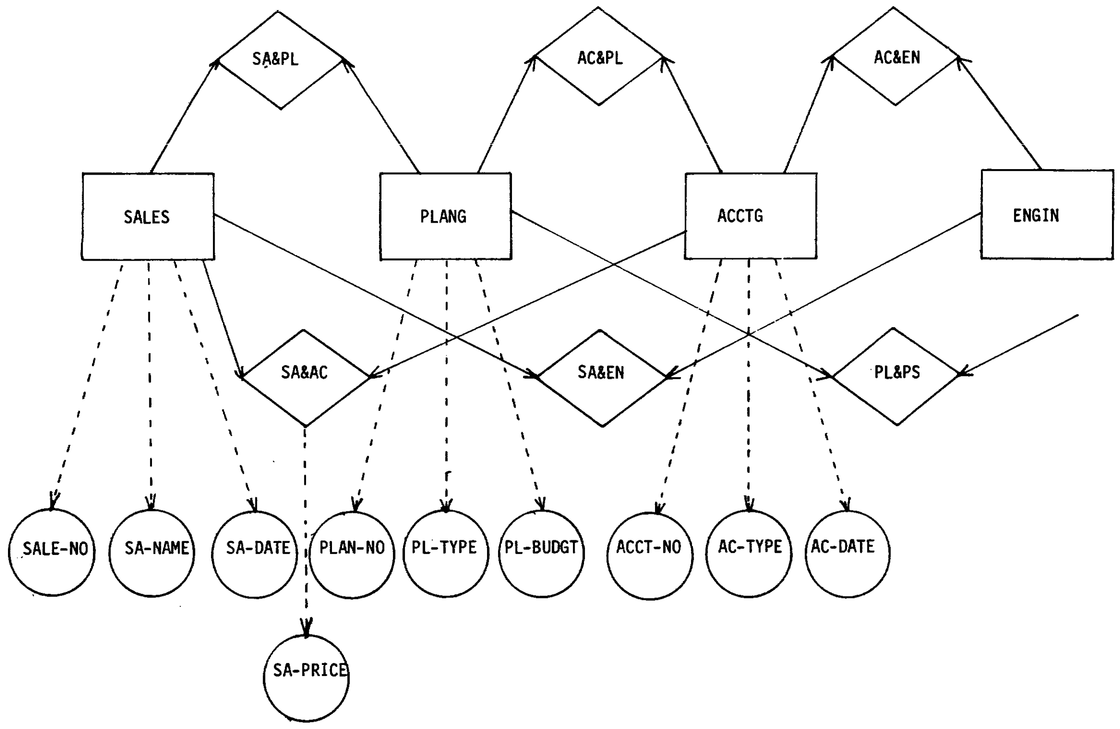


Figure 2(a)—Attributes and values of entity record types and relationship record types

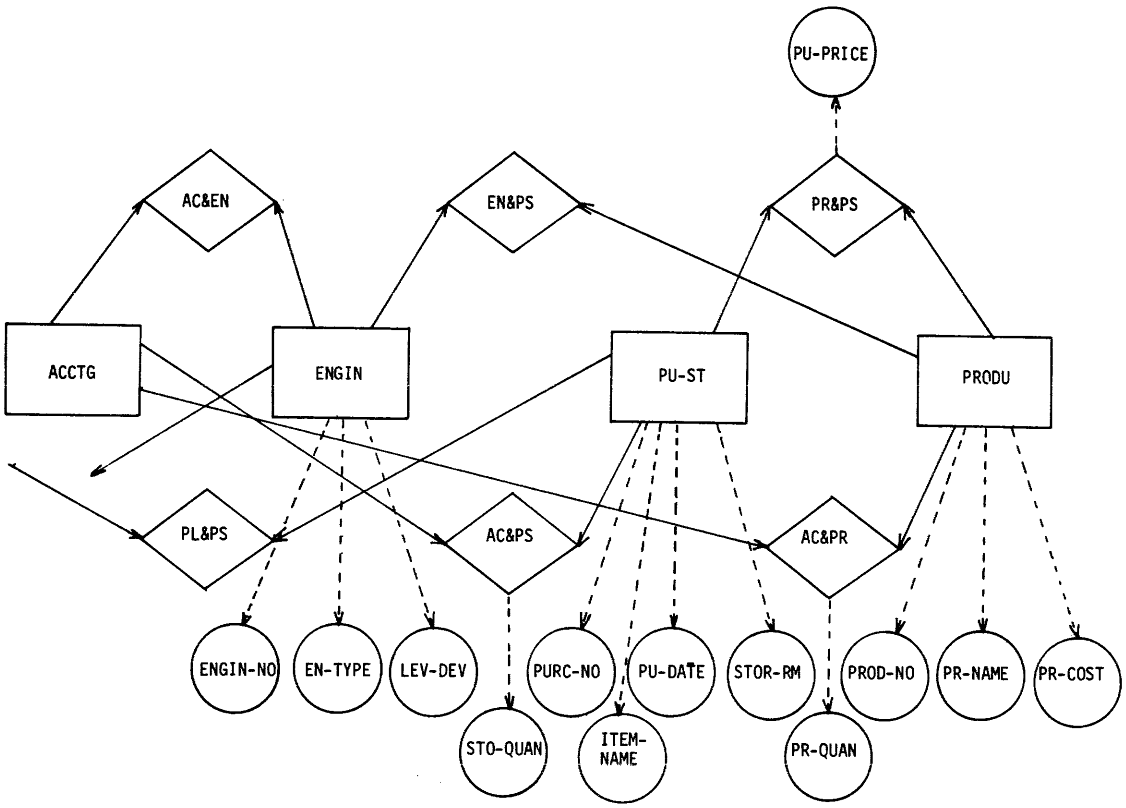


Figure 2(b)—Attributes and values of entity record types and relationship record types

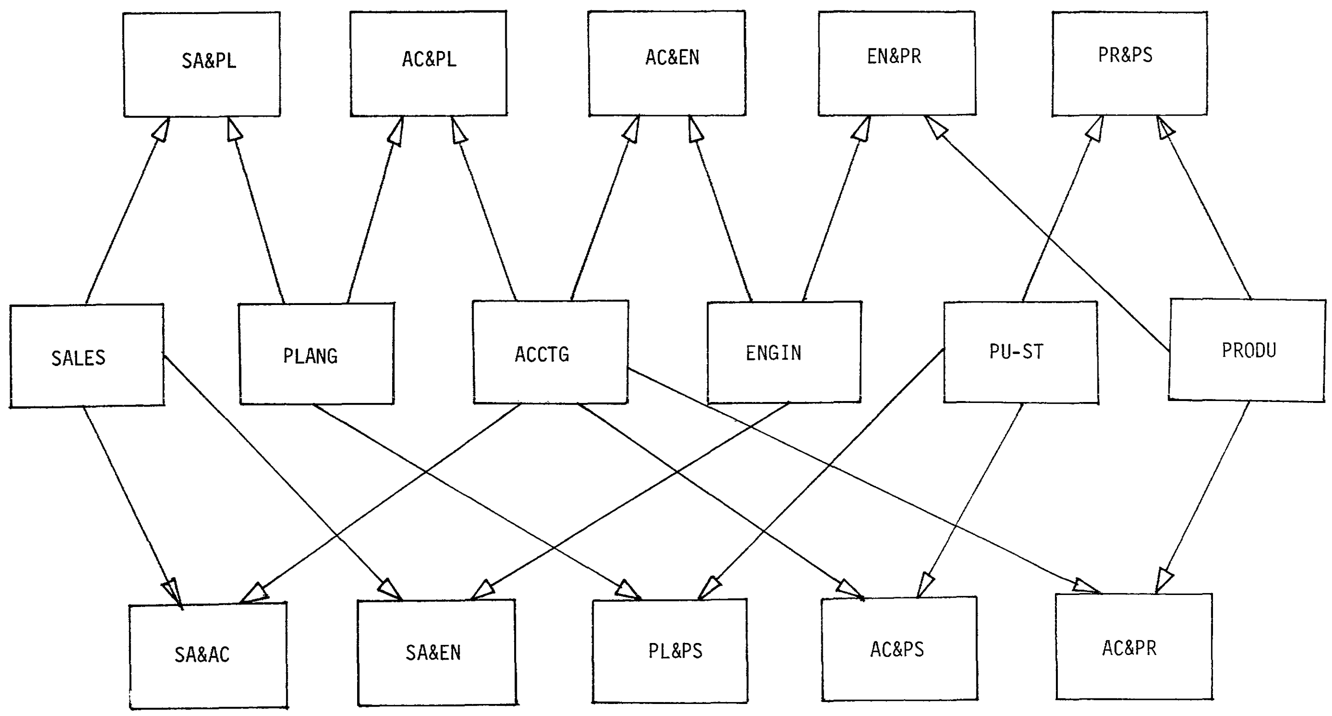


Figure 3—The data-structure diagram (transferred from the E/R diagram in Figure 1)

Sales record

SALES

SALE-NO	SA-NAME	SA-DATE

Planning record

PLANG

PLAN-NO	PL-TYPE	PL-BUDGT

Accounting record

ACCTG

ACCT-NO	AC-TYPE	AC-DATE

Engineering record

ENGIN

ENGI-NO	EN-TPEE	LEV-DEV

Purchase and stores record

PU-ST

PURC-NO	ITEM-NAME	PU-DATE	STOR-RM

Product record

PRODU

PROD-NO	PR-NAME	PR-COST

Figure 4—Record formats of entity record types

Sales and Planning Relationship Record	SA&PL	SALE-NO	PLAN-NO	
Sales and Accounting Relationship Record	SA&AC	SALE-NO	ACCT-NO	SA-PRICE
Sales and Engineering Relationship Record	SA&EN	SALE-NO	ENGI-NO	
Accounting and Planning Relationship Record	AC&PL	ACCT-NO	PLAN-NO	
Accounting and Engineering Relationship Record	AC&EN	ACCT-NO	ENGI-NO	
Accounting and Purchasing-Stores Relationship Record	AC&PS	ACCT-NO	PURC-NO	STO-QUAN
Accounting and Product Relationship Record	AC&PR	ACCT-NO	PROD-NO	PR-QUAN
Planning and Purchasing-Stores Relationship Record	PL&PS	PLAN-NO	PURC-NO	
Engineering and Product Relationship Record	EN&PR	ENGI-NO	PRDU-NO	
Product and Purchasing-Stores Relationship Record	PR&PS	PROD-NO	PURC-NO	PU-PRICE

Figure 5—Record formats of relationship record types (many-many relationships)

systems, this fixed format might not be appropriate because the environment for a decision-maker is volatile. In order to meet the unexpected situation a decision-maker may face, empty spaces may be left in the record as long as the relative position in the record is known. Since uncertainty is often the case, the flexibility in design is necessary for an efficient DSS. This paper emphasizes conceptual aspects of database design. Physical aspects are dealt with by Chen,[10] Date,[17] and Ullman.[47]

Features of E/R Approach

In logical database design, we first talk to the decision-makers or analyze the information requirements for various decisions. Then we combine these local conceptual models (or we may call them external models) into an integrated global conceptual model. After identifying all the entities and relationships, we draw an entity-relationship diagram. Next we identify the attributes and value types of all the entities and relationships and translate the E/R diagram into a data-structure diagram. Finally record formats are designed in accordance with the attributes and value types identified.

Because the record formats are equivalent to the relation scheme in the relational data model (if all the record formats, or relation schemes, are loaded with actual values), it becomes a database. Though all the record formats are very close to the normalized relations, we might go ahead to split or merge all the relations (traditionally, they are called files) through the normalization process as the situation warrants.† (Detailed discussion of normalization is beyond the scope of this paper; please see the above references.)

It is clear that the E/R approach is flexible and close to human thinking. If some particular relationship types changed from, e.g., one-many to many-many, a relationship record type is simply added, and the program or the database does not have to be changed. If the one-many relationships are already translated into relationship record types, even this process can be omitted.

The most important feature is that the E/R diagram can be translated into many different data structure diagrams to meet the needs of different data-processing environments. For example, it can be translated into relational, hierarchical, and

†See References 5, 6, 15, 17, 21, 30, 33, 45–48, and 51.

network data structure without any difficulties.[‡] All the entity records and relationship records are in the form of universal relations that can be immediately and conveniently used in a relational data model, logically or physically.[19,25,47,51]

Translation from the Global E/R Diagram into Logical Data Models

Translation from the global E/R diagram into hierarchical data structures is easy. The entity record types of the E/R approach are equivalent to the segment types of the hierarchical data model. First the root segments should be identified as the starting points and then, based on the conceptual and semantic meaning, the hierarchical trees can be structured. Through pointer segments, the many-many relationships can be handled. The most important features in the hierarchical data structure are that the logical parent pointer or the logical child pointer can be used for integrating several hierarchical organizations into a unified one, and by virtual storage, physical redundancies can be avoided. The set concepts are implied through the parent-child relationships.

Translation from the global E/R diagram into network data structures is also very simple. The entity record types of the global E/R diagram are equivalent to the DBTG record types. The relationship record types are equivalent to the DBTG set types. The set mechanisms are the most important concept and can be formally declared and used as access paths and navigation routes. Many owner records can be accessed through hashing mechanisms or other methods. Any member records can also be set up as the entry point if that is desired.

Translation from the global E/R diagram into relational data structures is straightforward.[35] The entity record types are translated into relations (tables). The major attribute or the combination of attributes that can be used to uniquely identify an entity (or a record) are selected as the primary key of the record. Every record must have a unique primary key for identification of the record. Sometimes, a foreign key that is the primary key value of some tuple in some other relation can be used to help identify the record involved.[17] The relationship record types are also translated into relations. The primary keys of the entities involved in the relationship are used as the primary key. If the relation is not in a simple form (e.g., including repeating groups in the relation), or all the non-primary attributes of a relation are not fully dependent on its primary key, or there exists transitive dependencies or multivalued dependencies, a normalization process should be used to produce the 4NF relations.

Toward A Distributed DBMS Under A Unified Approach

Volatile environment

Though there is no real DBMS built upon a unified data model concept yet, a theoretical framework has been established. Besides, such a unified approach is desirable for meeting the diversified requirements of information. A DSS is in a situation that demands diversified capabilities, and at the same time, the DSS designed must be very stable. It is closer to human thinking or natural human languages than conventional data models; the database management systems designed in accordance with the E/R diagram are more suitable for higher level user interfaces, and this is a very valuable feature for a DSS.

The distributed database

In addition to the above features, the database designed under the E/R approach can be logically integrated but physically distributed. Data distribution will be by clusters. Each cluster is organized in accordance with the guidelines of the E/R model and stored wholly at a single computer site. The system is a network that supports one logically integrated database under one data model and allows the user to enter repetitive transaction processing or ad hoc queries at any computer site. A user may access a part of the database, which may be stored at a remote site, and the system will connect the user to the portion of the database.[9,31] A set of standard functions has to be provided to the user, and it can be activated in a uniform manner by all the users. Every DBMS connected on the network is actually a unified and standard DBMS. It is called, according to Gardarin, a virtual DBMS.[22]

THE USER, THE DATABASE, AND THE MODEL INTERFACES

So far, the foundation has been laid for the design of databases that can support decision makers. A decision maker needs two capabilities: data management and modeling. The difference between a conventional DBMS and a DBMS that can support decision makers in their decision-making processes is that the latter DBMS has to incorporate modeling-handling capability and online interface with the database or databases. The model logic and its coefficients can be stored in a table (or relations in the relational model term) derived from the definition language and updated in the database.[42] Models draw data from the database and generate additional data placed in the database. According to GPLAN/DMS and GPLAN/QS, the GPLAN System[23,24] is built upon a DBMS that is composed of nine basic components. The most important component is the control program, which acts as an interface among the query system, the data management system, and the application program library. Working on models, the user sometimes creates records stored in the working storage space, and these records can be saved by issuing SAVE command. The entire system has been written in FORTRAN IV.

Bonczek[7] indicates that the GPLAN DML is used within the framework of a host language (e.g., FORTRAN, COBOL). Each command in the DML is translated into a subroutine call. These subroutines extend the FORTRAN language to a complete data manipulation language. These network-oriented DML are inferior to the high-level query

[‡]See References 1, 2, 10, 19, 25, 32, 36, and 39.

language[9] that provides interfaces among the users, models, and databases. This query language permits selective retrieval, update, and entry of data from or into databases. It also provides standard or nonstandard reports.

SLIM,[16] a management simulation system for teaching MIS and DSS, is also extended to FORTRAN IV. The SLIM system consists of a dynamic database and data dictionary, a simple query language and query processor, and a module for database administration. It includes the essential elements of the query portion of a DSS. The batch-oriented commands provide a rudimentary data manipulation language with BASIC-like syntax. It is really the beginning of representative software for managerial decision support systems.

Some systems extend to APL, PASCAL, PL/1, etc.—higher level languages. Stohr[44] extended APL in the planning system using the EDBS[20] (database management system). It is designed to support operation research models in the context of an integrated planning system. The database management system is used to keep track of the complex information flows between decision-making units and computer programs. It has four APL files: *The Data Dictionary File,* containing the definition of models, processes including input/output variables, and problem statement definition as well as report definitions; *The Database Directory* for maintaining logical relationships between models and processes; *The Data File,* containing the actual input/output data values and character matrices; and *The Function Library,* containing the code for all functions used by the planning system. All functions must be registered in the function library before they can be used by a process. Keen[28] and Sprague[42] also discussed using APL as extended language. Detailed discussion is beyond the scope of this paper.

CONCLUSION

Designing a database-oriented DSS represents a new challenge, which requires integrating computer technology, management science, and organizational theory. This paper has laid the foundation for such an attempt. The design criterion for such a comprehensive system is that it must handle the interfaces among the users, databases, and models. We have reviewed the database design basics. The unified approach is the goal, and the entity-relationship model is the best conceptual model that can lead to a global conceptual enterprise view of data. Various translation processes from the E/R diagram into the data-structure model have been discussed.

Though there is no unified approach for the design of databases for a DSS, the theoretical foundation has been established. A comprehensive DSS that can support a decision maker in standard or nonstandard reports, batch or online processing, is within reach.

An example of a unified approach has been illustrated. Some existing DSSs that extend to the host languages also have been examined briefly. Following this framework, a genuine DSS that can support a decision maker in his structured, semi-structured, or unstructured problem solving is not only desirable, but also possible, for implementation in a managerial context.

REFERENCES

1. Adiba, M., C. Delobel, and M. Leonard. "A Unified Approach for Modeling Data in Logical Data Base Design." In G. M. Nijssen (ed.), *Modeling in Data Base Management Systems.* Amsterdam: North-Holland, 1976.
2. Adiba, M., and C. Delobel. "The Problem of the Cooperation Between Different D.B.M.S." In G. M. Nijssen (ed.). *Architecture and Models in Data Base Management Systems.* Amsterdam: North-Holland, 1977.
3. Alter, Steven L. "Decision Support Systems: Current Practice and Continuing Changes." Addison-Wesley Series on Decision Support. Reading, Mass.: Addison-Wesley, 1980.
4. "ANSI-X3-SPARC/DBMS, Study Group Report." Washington, D.C.: American National Standards Institute, 1975.
5. Beeri, Catriel. "On the Membership Problem for Functional and Multivalued Dependencies in Relational Databases." *ACM Transactions on Database Systems,* 5, (1980).
6. Bernstein, Philip A. "Synthesizing Third Normal Form Relations from Functional Dependencies." *ACM Transactions on Database Systems,* 1 (1976).
7. Bonczek, Robert H., et al. "Aiding Decision Makers with A Generalized Data Base Management System: An Application to Inventory Management." *Decision Sciences,* 9 (1978).
8. Bonczek, Robert H., Clyde W. Holsapple, and Andrew B. Whinston. "The Evolving Roles of Models in Decision Support Systems." *Decision Sciences,* 11, No. 2 (1980).
9. Chamberlin, D. D., et al. "Support for Repetitive Transactions and Ad Hoc Queries in System R." *ACM Transactions on Database Systems,* 6 (1981).
10. Chen, Peter Pin-shan. "The Entity-Relationship Model—Toward a Unified View of Data." *ACM Transactions on Database Systems,* 1 (1976).
11. Chen, Peter Pin-shan. "The Entity-Relationship Approach to Logical Data Base Design." Q.E.D. Monograph Series, No. 6, 1977, Q.E.D. Information Sciences, Inc., 141 Linden St., Wellesley, MA 02181.
12. "CODASYL Data Base Task Group Report, 1971." CODASYL, Association for Computing Machinery, New York, April 1971.
13. CODASYL Data Description Language Committee, Proposed Revision of the 1971 DBTG Report, February 1973.
14. CODASYL Data Base Language Task Group, Proposed Revisions of the 1971 DBTG Report, June 1973.
15. Codd, E. F. "A Relational Model of Data For Large Shared Data Banks." *Communications of the Association for Computing Machinery,* 13 (1970).
16. Courtney, James F., et al. "SLIM: A Management Simulation System for Teaching MIS and DSS," *Interface,* The Computer Education Quarterly, Vol. 2/Issue 2 (1980).
17. Date, C. J. "An Introduction to Database Systems." 3rd ed. Reading, Mass.: Addison-Wesley, 1981.
18. De, Prabuddha, and Arun Sen. "Logical Data Base Design in Decision Support Systems." *Journal of Systems Management* (1981).
19. Deen, S. M. "A Canonical Schema for a Generalized Data Model with Local Interfaces." *The Computer Journal,* 23, No. 3 (1980).
20. "Education Data Base System," User's Guide. Department of Computer Services, the University of Calgary, Alberta, Canada, September 1977.
21. Fagin, Ronald. "Multivalued Dependencies and a New Normal Form for Relational Databases." *ACM Transactions on Database Systems,* 2 (1977).
22. Gardarin, G., et al. "An Approach Towards a Virtual Data Base Protocol for Computer Networks." *Proceedings, AICA,* 1977.
23. Haseman, W. D., et al. "Automatic Application Program Interface to A Data Base." *The Computer Journal,* May 1977.
24. Haseman, William D. "GPLAN: An Operational DSS." School of Urban and Public Affairs, Carnegie-Mellon University, Pittsburgh, Pa., 1977.
25. Kalinichenko, L. A. "Relational-Network Data Structure Mapping." In G. M. Nijssen (ed.), *Modeling in Data Base Management Systems.* Amsterdam: North-Holland, 1976.
26. Keen, Peter G. W. and Michael S. Scott Morton. "Decision Support Systems: An Organizational Perspective." Addison-Wesley Series on Decision Support. Reading, Mass.: Addison-Wesley, 1978.
27. Keen, Peter G. W. "Decision Support Systems: Translating Analytic Techniques into Useful Tools." *Sloan Management Review,* 21 (Spring 1980).
28. Keen, Peter G. W., et al. "Building a Decision Support System: The Mythical Man-Month Revisited." In J. F. Bennett (ed.), *Building Decision Support Systems.* Addison-Wesley Series on Decision Support. Reading, Mass.: Addison-Wesley, 1980.

29. Kerschberg, L., A. Klug, and D. Tsichritzis. "A Taxonomy of Data Models." In P. C. Lockemann and F. J. Neuhold (eds.), *Systems for Large Data Bases.* Amsterdam: North-Holland Publishing Company, 1976.
30. Lee, D. T., et al. "Quintary Tree: A File Structure for Multidimensional Database Systems." *ACM Transactions on Database Systems,* 5 (1980).
31. Lien, Edmund Y., et al. "Design of a Distributed Entity-Relationship Database system." *Proceedings of COMPSAC,* 1978.
32. Line, Y. Edmund. "Hierarchical Schema for Relational Databases." *ACM Transactions on Database Systems,* Vol. 6, No. 1, March 1981.
33. Ling, Tok-Wang, et al. "An Improved Third Normal Form for Relational Databases." *ACM Transactions on Database Systems,* 6 (1981).
34. Martin, James. *Computer Data-Base Organization* (2nd ed). Englewood Cliffs, N.J.: Prentice-Hall, 1977.
35. McCarthy, W. E. "An Entity-Relationship View of Accounting Models." *The Accounting Review,* 54, No. 4 (1979).
36. Mercz, Iaszlo I. "Issues in Building a Relational Interface on A CODASYL DBMS." In Braechi/Nijssen (eds.), *Data Base Architecture.* Amsterdam: North-Holland, 1979.
37. Michaels, Ann S. "A Comparison of the Relational and CODASYL Approaches to Data-Base Management." *ACM Computing Surveys* 8 (1976).
38. Morton, Scott M. S. "Management Decision Systems: Computer Based Support for Decision Making." Division of Research, Harvard University, 1971.
39. Nahouraii, E., L. O. Brooks, and A. F. Cardenas. "An Approach to Data Communication Between Different Generalized Data Base Management Systems." In P. C. Lockemann and E. J. Neuhold (eds.), *Systems for Large Data Bases.* Amsterdam: North-Holland, 1976.
40. Olle, T. William. *The CODASYL Approach to Data Base Management.* New York: John Wiley & Sons, 1978.
41. Sprague, R. H., Jr., and H. J. Watson. "MIS Concepts—Part II." *Journal of Systems Management,* 26, No. 2 (1975).
42. Sprague, Ralph H., Jr., et al. "A Decision Support System for Banks." *Omega—The International Journal of Management Science,* 4, No. 6 (1976).
43. Sprague, Ralph H. "A Framework for the Development of Decision Support Systems." *MIS Quarterly,* December 1980.
44. Stohr, Edward A., et al. "A Database for Operations Research Models." *International Journal of Policy Analysis and Information Systems,* 4, No. 1 (1980).
45. Su, Stanley Y. W. "A Semantic Association Model for Conceptual Database Design." In Peter P. Chen (ed.), *Entity-Relationship Approach to Systems Analysis and Design.* Amsterdam: North-Holland, 1980.
46. Su, Stanley Y. W., Herman Lam, and Der Her Lo. "Transformation of Data Traversals and Operations in Application Programs to Account for Semantic Changes of Databases." *ACM Transactions on Database Systems,* 6 (1981).
47. Ullman, Jeffrey D. *Principles of Database Systems.* Computer Science Press, 1980.
48. Vetter, M. and R. N. Maddison. *Database Design Methodology.* Englewood Cliffs, N.J.: Prentice-Hall, 1981.
49. Wagner, G. R. "Decision Support Systems: The Real Substance." *Interfaces,* 11, No. 2 (1981).
50. Zaniolo, Carlo. "Multimodel External Schemas for CODASYL Data Base Management Systems." In Bracchi/Nijssen (eds.), *Data Base Architecture.* Amsterdam: North-Holland Publishing Company, 1979.
51. Zaniolo, Carlo. "On the Design of Relational Database Schemata." *ACM Transactions on Database Systems,* 6 (1981).

HARDWARE

Richard Lampman
Hewlett-Packard Company
Cupertino, California

The steady reduction in cost per function for hardware continues to be the primary force in expanding the role of information processing. The improvements in price and performance have brought the technology across a threshold; it has become an affordable commodity for individuals.

The papers in this track, which focus on new designs and their uses, fall into four groups:

1. Increasing speed and new applications for high-speed hardware
2. New inexpensive hardware and the implications of this less expensive hardware
3. The use of the new inexpensive hardware to build multiprocessor systems
4. The emerging information technology built around videotex

Systems designed for high-speed computation are one use of the price/performance-improved hardware. Applications examples are discussed in one set of papers to illustrate the expanding applications. Another set of papers addresses specialized hardware that can be dynamically reconfigured to optimize specialized computations. The ongoing research in multiprocessors for parallel processing is discussed in another set of papers.

Not only has price/performance improved; the entry price of systems has continued to decline. The availability of standardized hardware and peripherals has changed the structure of the inexpensive-systems market; two sessions discuss the implications of this.

The use of multiprocessor systems is beginning to accelerate rapidly, driven by the improved price/performance that can be obtained and by the increasing need for fault-tolerant computers. Two sessions explore the design and application of multiprocessor systems.

Videotex, a new technology that is just arriving in the United States, will make some forms of information widely available at lower cost. The session on videotex will explain the technology and its implications.

Although much stress is being placed on the applications of computers, it is the hardware foundation that determines the economics of computer use. The hardware sessions show how new hardware continues to propel computer use into new applications.

Universities and the future of high-performance computing technology

by KENNETH G. WILSON
Cornell University
Ithaca, New York

ABSTRACT

Universities have four main roles in the growth of computer technology. (1) Universities provide trained personnel for both the computing industry and industrial users of computing. (2) Universities carry out the majority of very advanced software research and prototype development. (3) Universities can provide an advance market for new computing hardware and can help manufacturers refine innovative products to serve commercial markets. (4) Universities carry out basic research needed to support continued technological growth. There are extraordinary developments taking place in computing that it is difficult for U.S. industry to absorb without the universities' help. Present university support is grossly inadequate for any of these functions. I propose a computer tax as a model of how universities should be supported. It would provide the hundreds of millions of dollars needed to restore the universities' role in computing-technology growth. More importantly, it would ensure that universities' future growth in technological areas is at the same rate as that of computing technology itself.

A. INTRODUCTION

In the coming decade, large-scale computer simulation will play a rapidly growing role in industrial and governmental research and development. Computer simulation is needed especially in connection with the search for and processing of energy resources, in connection with industrial or natural processes involving fluid flow, study of new industrial materials at microscopic or molecular levels, and so on. The pressures of changing raw materials, environmental constraints, international competition, and decreasing computing costs will lead to a rapid increase in the use of computer simulation to replace experimentation and prototyping or to provide information not otherwise accessible. Computer simulation, if feasible at all, will usually be cheaper and faster than any alternatives, and can reduce the total time and cost of research and development projects.

Major changes are needed to support a rapid growth of computer simulation. Basic theoretical science research is evolving new methods (such as the "renormalization group"[1]) that can greatly extend the capabilities of scientific simulation, but only if a very strong program of computing support, research and training is established in universities; at present such programs are haphazard, with grossly inadequate computing resources. The most powerful computers commercially available or announced (including supercomputers such as the Cray-1 or forthcoming Cray-2) are themselves inadequate by many orders of magnitude to meet future computer-simulation needs. The extraordinary capabilities of current computing technology must be pushed to their utmost extent in radical, highly parallel architectures to make possible computing systems thousands or even millions of times more powerful than today's supercomputers, if the full potential of computer simulation is to be realized.

Universities must have a strong role in developing the computers that are to be used for computer simulation. The benefits to industry of university involvement in computer development will become extremely important and complement the benefits that will be obtained from university research and training in areas using simulation. This paper describes the need for university involvement in computer technology development, and outlines a framework for this involvement. This paper concludes with a controversial proposal for financing the universities' role in computer-related technology.

In this paper I describe the characteristics of the large-scale computing market and how it blocks innovation. I propose a multistage design process for new large-scale computing systems, with universities providing a market and test and demonstration sites for early stage designs. The critical role of universities in advanced-software development and testing will be stressed.

To finance the university market, I propose a use tax, namely a sales tax on computing equipment.

B. BARRIERS TO INNOVATION IN THE U.S.

The barriers to innovation in large-scale computing are described in Sections B.1 to B.3. Some further comments on the large-scale-computing scene are given in Section B.4.

B.1 Software Compatibility

Most mainframes bought in the U.S. are required to be totally software compatible with existing systems. This requirement is imposed to maintain the viability of existing software, which represents a major investment and is virtually impossible to convert for new computer architectures. IBM has a major research and development effort in Josephson junction technology, in the hope that it will provide a long-term growth path for their 360-370 mainframe architecture. But there seems to be no hope that this technology, even when fully developed, can make the 370 architecture competitive with future, more novel architectures.[2] The prospects are no more promising for other current mainframes.

Cray Research has carved out a small niche for itself supplying supercomputers to customers willing to sacrifice software compatibility in return for the somewhat greater power of the Cray-1. Cray Research has made enough profit to be able to work on designs for the Cray-2, a supercomputer which is projected to be about ten times more powerful than the Cray-1.[3] However, even the Cray-2 will be far short of the capabilities of parallel systems, and there seems to be little hope that further development of the Cray architecture can compete with parallelism.

There is one U.S. project that is a natural starting point for investigating parallelism specifically for scientific simulation. This is the Denelcor HEP supercomputer, with initial deliveries scheduled for the spring of 1982.[4] The difficulties faced by this project are extreme: at the moment not even a Cray-sized market is open to it. Cray Research was able to get started by supplying early production models to the Los Alamos and Livermore Laboratories. Now Los Alamos and Livermore have all but ceased to provide a market for unique new computers; their budget goes almost entirely for the purchase of Cray-1's. This does not mean that they find the Cray-1 totally satisfactory. Not even fifty Cray-1's would provide enough computing power to enable Livermore and Los Alamos to carry out their assigned missions.[5] However, Livermore and Los Alamos now demand either total software compatibility with the Cray-1 or a very substantial advance over the Cray-1. The HEP presently meets neither requirement. Thus the problem for Denelcor is how to win initial acceptance for its product. The HEP is a serious supercomputer effort but its speed is not overwhelming. Software development for the HEP is only just getting underway. Its greatest strength right now is its large physical-memory capacity (up to 1 *gigabyte*), but the cost of this memory is currently very high.

There are a number of large-scale systems, some still in development, aimed at special purpose markets. These include the Illiac IV,[6] Goodyear's Massively Parallel Processor,[7] ICL's Distributed Array Processor (DAP),[6] and Control Data Corporation's Advanced Flexible Processor.[8] There are also a number of experimental noncommercial projects.* At present these systems serve only a very limited fraction of the large-scale market. Their potential is rarely explored fully because they lack a full complement of software. They can be highly innovative in their initial design, but it is not easy to keep them technologically current over long periods of time.

A crucial need is to have a much greater variety of large-scale systems entering the general large-scale market. I will stress the Denelcor system because it *is* entering this market and deserves a fair hearing in this market. My proposal, described later, will allow many other large-scale ventures to form and also receive a fair hearing. I endorse parallelism in this paper only as an opportunity that needs to be explored; my proposal would allow many other options to be explored too.

B.2 The Law of Diminishing Returns

There is a law of diminishing returns in large-scale computing. It states that if a single computing task cannot be carried out on a current mainframe, then a factor-of-2 increase in power is useless for the task, and a factor-of-1000 increase in computing power is as likely to be required as a factor-of-10. The reason for this law is that tasks that would benefit from a factor-of-2 increase in computing power rarely require more than a minicomputer to carry out—such tasks are extremely unlikely to challenge a mainframe. The typical consequence of the law of diminishing returns is that upgrades from mainframes or large minicomputers may need to be by factors of 10, not 2, and that over a decade several factor-of-10 upgrades could be necessary. Parallel architectures may allow this level of upgrading. Another consequence of the law of diminishing returns is that an upgrade by a factor of ten does not justify a factor-of-10 increase in cost.

The law of diminishing returns does not always apply. Consider two examples. A large computing center, presently running 1200 jobs a day, needs to expand to accommodate 2400 jobs a day. The law of diminishing returns does not apply. A factor-of-2 increase in power is fine, and the computing center can easily afford to double its cost. The second example is a small research group that has filled up a superminicomputer. Typically this occurs because of individual jobs that are using whole days of computer time. An upgrade to two superminis is useless for the reasons given above.

*For example, the SI project at Lawrence Livermore Laboratory.

Over the next five to 10 years, innumerable small research and development groups, as well as individual engineers and other professionals, will have learned how to use supermini-level systems in a personal computing mode. They will have learned how to use days of computing time at a stretch—productively. (Computer users who use a centralized facility are typically severely upbraided for submitting day-long jobs, either because it disrupts other users or because computing time is *expensive*!) They will be ready for upgrades; and they will be face to face with the law of diminishing returns. The needs of centralized facilities to double their job throughput is likely to become a considerably less significant factor in the large-scale computing market than it is now.

B.3 The Software Barrier

New large-scale computing systems do not instantly deliver full value to their users. They are not like new automobiles, which one can drive off after a few minutes spent locating the controls. Years elapse between the first delivery of a major system built around a new architecture and the full production use of the system. This was true of the CDC 6600 and 7600, the IBM 360 system, and the Cray-1. Intervening years are spent building systems software, converting application software packages for the new system, developing new applications software, and preparing many volumes of documentation for all three kinds of software. Time is also needed to train maintenance personnel in both hardware and software, to establish applications areas that are suited to the new system, and to train users in procedures making effective use of the new system. An enormous user base is needed to encourage this software and maintenance buildup. An even longer period is needed to bring all the software to full working order and into full compliance with the documentation; only heavy use by thousands of customer groups can adequately exercise complex mainframe software, thereby establishing some confidence in its full reliability. The IBM software for the 360-370 series is still evolving, 14 years after its introduction; new releases of basic 360-370 systems software occur every 3 months.† The documentation for the 360-370 series fills an entire library bookshelf, from floor to ceiling.

In the early years of a new system, the lack of software seriously limits its usefulness; the software that does exist may fail frequently. Documentation is nonexistent, out of date, or wrong. Intense user exasperation results, because each problem that arises, no matter how trivial, may take months to resolve. Each problem could be the fault in a user's program, or in a system program, or in the hardware, or just in a manual. Tracing down the problem and then fixing it may require intense study of all four possible sources of the difficulty by highly experienced personnel.

Commercial and governmental customers are not (and should not be) very eager to try out a new system until these early years are past and smooth operation of both hardware and software has been demonstrated.

The software barrier becomes worse when one contem-

†There are about ten releases per year.

plates systems with novel architectures having much greater power than the Cray-1. These architectures can themselves pose major headaches for software developers, and the increase in computing power is usually accompanied by greater complexity in applications.

B.4 Networking Problems

The capabilities of microprocessor chips are now increasing very rapidly, and plans are being established to allow multiple microprocessor systems to be linked together through local networks such as Ethernet.‡ One might hope that further development of these systems would lead to systems with the power and expansion capability that the future requires. Unfortunately, the networks planned in the U.S. are many orders of magnitude too slow to allow a single large computation to be divided among many processors on a network; that would require far more communication traffic between the processors than the networks can handle.

The Denelcor system is designed around a very high speed, expandable network hundreds of times faster than Ethernet.[9] The Denelcor network links from one to 16 processors to an arbitrary number of independent memory modules. The aggregate data-transfer rate of the Denelcor network increases as the number of processors increases, so that communications delays should not occur and cause bottlenecks. The special-purpose high-speed processors also contain high-speed switches or communication paths.

Future multipurpose very-large-scale systems are likely to be judged mainly by their aggregate internal data transfer rates and their aggregate memory-storage capacity. A number of massively parallel network designs are currently under study by computer scientists.[10] These designs could provide enormous aggregate data-transfer rates. The Denelcor HEP network is a harbinger for some of these network designs, and the large memory capacity of the HEP is a hint of the much larger memory capacities to come.

C. UNIVERSITIES AND A MULTISTAGE DESIGN PROCESS

C.1 Preliminaries

In this section I will describe in detail the role of universities in providing an advance market for new large-scale computing systems. I will explain this role in the context of a multistage design process for new hardware and software. The multistage design process I advocate is well suited to the U.S. university and industrial scene and will allow U.S. industry to compete aggressively in world markets despite high interest rates and short-term business-management pressures. In the multistage model, each design stage for a new computing system is short and market driven, with heavy competition encouraged. The early stages of major new computing systems would usually be too primitive for commercial markets; thus universities would be the primary market for the early stages of systems. For many reasons, explained below, universities can obtain useful work from a new computing system before it is ready for commercial use, and for just as many reasons it is economically important that universities have a strong early involvement with these new systems.

The multistage design process I will describe is well supported by past experience. The important computing systems of today have long histories of prior models. For example, the IBM 3081 series, just getting underway, was preceded by the 3000 series, the 370 series, and the 360 series computers. The VAX of Digital Equipment Corporation follows many models in the PDP-11 line. Universities have always provided an important early market for new companies starting a new product line. Digital Equipment, Prime Computer, and Amdahl are just three examples of companies whose first annual reports featured university customers.

The proposal I will outline is based on an example of a multistage computer design process that I am personally familiar with. The example is the design history of the Floating Point Systems FPS-164 Attached Processor.[11] Deliveries of the FPS-164 have only just begun, so it will not be surprising if readers have not heard of it or of Floating Point Systems. I will review this history because many details of my proposal are illustrated by it.

C.2 The FPS-164 Design History

The first stage of the design process was a custom-built special-purpose processor called the AP-120, with no floating-point capability, designed and built by Culler Harrison Inc. The second stage was the design of a floating-point processor called the AP-120B.[12] Floating Point Systems carried out this design in a very short time in 1975, using a number of the architectural ideas of the AP-120. At the time of the design, Floating Point Systems had only a handful of employees. The AP-120B was sold with no higher-level software. It was a spectacular advance in hardware. It was typically bought to be attached to an inexpensive minicomputer; it provided a hundredfold increase in speed over the minicomputer for roughly the minicomputer's price. Programming the AP-120B was difficult, but usually only a few small but time-consuming subroutines were converted for the AP-120B. As a result, the software development problems for the AP-120B were manageable. Floating Point Systems has been growing rapidly ever since, as sales of the AP-120B have grown.§

The third design stage was the design of the FPS-164 Attached Processor. A commercial verdict on its success is not yet in. However, the main features of the FPS-164 design process are known.[12] The FPS-164 is a further elaboration of the AP-120B architecture. The emphasis was on achieving broader functionality and higher reliability than the AP-120B rather than on any architectural advance. The FPS-164 offers greater numerical precision, more memory capacity, and more high-level software than the AP-120B, along with a

‡Local networks such as Ethernet are not designed for large-scale applications. However, local networks rather than high-speed networks are currently in vogue.

§Floating Point System's gross income was three and one-quarter million dollars in 1976; it was 58 million dollars in 1981.

modern remote diagnostic capability. However, it does not provide as spectacular a jump in capability or cost effectiveness over existing systems as the AP-120B did; for the FPS-164 this jump will be, I think, roughly a factor of 5 to 10 over competitively priced systems. The design of the FPS-164 reflects the need of FPS, which is now a medium-sized company, to provide a solid commercial product serving a relatively broad market.

The design of the FPS-164 was preceded by a public demonstration of the capability of the AP-120B architecture to support a broad range of scientific applications. This demonstration was carried out at Cornell University. The Cornell Computing Center, in collaboration with a consortium of scientific researchers with large-scale computing needs and very little money, bought an AP-190L array processor from FPS in 1977. It was attached to the central university computing system, an IBM 370-168 computer. The computing center sponsored the development of a FORTRAN compiler for the AP-190L, which enabled consortium members to use the AP-190L for a variety of scientific applications.

This software support was minimal. The documentation Cornell provided for the AP-190L was a single 96-page manual. The compiler was primitive, and it has provided lots of software headaches for users, especially in its early days. The main users of the AP-190L have been the consortium members who helped pay for it; they and their students have overcome the practical problems of using the AP-190L and a number of graduate student theses have resulted. The FPS-164 will be a much more satisfactory system for the pattern of usage Cornell pioneered on the AP-190L.

The Cornell example provided Floating Point Systems with a wealth of experiences of the advantages and weaknesses of the AP architecture for general scientific use. The Cornell example provides potential FPS-164 customers with a model for its use; the Cornell experience is likely to play an important role in marketing the 164. Cornell will be an early demonstration site for the FPS-164.

C.3 General Description of the Multistage Design Process

The multistage design process would involve an alternation between secret industrial computer-design efforts and open university test and demonstration projects that would show off the resulting systems. The industrial design stages would be short, partly because of the normal need to bring a product to market and make money on it, and partly to keep the designs current with rapidly advancing computing technology. Each new design stage would be an opportunity to incorporate more modern integrated-circuit technology into the design, to take advantage of the latest diagnostic and quality control procedures, and so on. The universities should also be under pressure to get new systems running quickly so that the manufacturer and potential commercial customers can get quick feedback on the benefits and problems the new systems provide.

Systems in the first design stage would mostly be for experimental use by a small community of users, with users and designers in frequent informal contact so that changes (even major ones) can be made quickly and a few phone calls can substitute for documentation. Subsequent design stages would take the most effective architectures and refine them, adding more hardware facilities, building up higher-level software, and in general exploiting the basic architectural framework for successively larger and broader markets. Universities would typically buy the first few production models of intermediate-stage designs for test and demonstration purposes, but much of the follow-on market for these systems would be commercial, for special-purpose applications. The final stages of the design process would involve industry giants like IBM preparing systems for world-wide distribution with multipurpose capabilities, proven software and full maintenance support. The last design stages would be concerned heavily with maximizing software compatibility with the manufacturer's previous product lines, as well as with the many detailed optimizations involved in serving a multi-billion dollar market.

The uses of the university demonstration systems should be as varied as possible in order to maximize the effectiveness of the demonstration. All kinds of basic research should be carried out. Theoretical physicsts, chemists, astrophysicists, and applied mathematicians would carry out simulations of real or fanciful physical situations. High-energy physicists, space scientists, and others would analyze enormous data streams from their experiments. Innumerable engineering, database, and information-retrieval applications should be pursued. Musicians, artists, humanists, and social scientists would have other uses to try. The relevance of any of this research would be unimportant. The only requirement would be breadth—a broad range of research activities would be needed to provide a broad range of tests for the new computing systems. It is important, however, that the computing systems tested be multipurpose systems rather than specific to the research being pursued.

It is important that the research carried out on the new computing systems be of high quality. The best researchers have the strongest motivation to stick with their projects despite the difficulties the new systems present. The best researchers will place the most strenuous demands on their equipment and are the ones most likely to discover new application areas for it. Above all, students working with the equipment need to see it being used effectively, so that they will graduate with a positive feeling about computing and what it can do.

Current policies that have the effect of diminishing the U.S. university research efforts must be reexamined. Many high-quality projects that are now being abandoned or deferred need to be fully restored so that they can help build the market for advanced computing equipment in the service of innovative and varied research.

Many systems would fail at various stages of the design effort. The market supporting innovation must be large enough to survive these failures, and must encourage only the most successful technologies through further design stages. The university projects that would be the primary market for products of early design stages would also have a high failure rate. The demands for innovation from the university demonstration projects would be severe, and only a small fraction of the university projects would be fully successful. The university efforts require discovering strengths and weaknesses of a new system despite very considerable barriers of instability and inadequate software, finding novel applications for the system that will broaden its commercial market, and developing the initial framework for the system's use. These are not easy tasks. The Cornell Array Processor project was a success, but in many ways we were lucky to succeed; no one could guarantee success for any specific university project that tried to be as bold as we were.

Through success or failure, a research group's normal functions of student training and basic research must continue. Other computing systems must always be available to back up the research effort; failure can be as important to learning as success is.

C.4 Benefits of the Multistage Plan

The model of a multistage design process with heavy involvement by universities is attractive in many ways. University research groups are well suited to working with new computing equipment. An abundance of graduate and undergraduate student labor eases the difficulties and frustrations of working with new equipment. Faculty members carrying out research on these systems often have a high level of expertise that enables them to overcome the difficulties of poor software and inadequate documentation. University research is often less constrained by pre-existing requirements and software than commercial or governmental computing. Researchers can adjust their research topics and directions to avoid the shortcomings of a new system and take advantage of its benefits. Students writing programs from scratch can adapt relatively easily to new architectures and new software development tools. The competitive character of university research causes both faculty and students to seek out new or unexpected applications of a new system in order to obtain a research advantage.

The training functions of university test and demonstration projects cannot be overemphasized. In an era of rapidly developing computer technology, technology that is already established when a student enters a university is on its way to obsolescence by the time the student graduates and enters the work force. Providing students with experience in computing technology in advance of its commercialization is the only way for them to be reasonably current at graduation. New systems, by virtue of their instability and poor software support, can be the source of meaningful part-time jobs for many undergraduates.

A university test-and-demonstration project for a new commercial product is a superb framework for technology transfer from universities to industry. University researchers working with a new piece of equipment are strongly motivated to find ways to enhance the equipment to make it more effective for their research. Enhancements to computer equipment are likely to be especially helpful to a broad class of users of the equipment, because of the inherent flexibility of computers. A small company or a new product group which is trying to win acceptance for a new product has a strong motivation to listen to the ideas of the university users, especially if the success of the university demonstration project is crucial to the future of the product. Interaction between university researchers and an industrial design group is much easier and more effective at a very early stage in the design process, when major changes can be made easily and university personnel are working on a personal basis with the design group. Frequent interchanges take place as the researchers encounter difficulties and seek the help of the designers in helping with these difficulties.

The improvements developed in the course of a university test operation may seem minor in the early phases of a new product. However, if the product develops to the point of serving a multi-billion-dollar market, the economic impact of the university's efforts can be profound. In contrast, once a product is in the final stages of preparation for a billion-dollar market, a massive bureaucracy is in place surrounding product design, development, and marketing and any effective influence by a few university researchers is impossible.

The university test and demonstration projects would develop strong ties with potential commercial customers for new systems. Cornell has fielded many inquiries from potential commercial customers for the AP-190L or the FPS-164. We accept the site visits, and play a strong role in the Floating Point Systems user's group; quite a number of papers by Cornell authors have come out of the user's group meetings.[14–19] At user-group meetings, university and commercial users of Floating Point Systems get together to exchange experiences and software, thus establishing another university-industry link where technology transfer can take place.

D. UNIVERSITIES AND THE SOFTWARE PROBLEM

A very important function of the universities is to attack the software complexity problem. Computer science departments are already devoting a major part of their research effort to the search for better methods of software development. Many promising developments have occurred in the last few years; most of them are still in the preliminary prototyping stage. The old view that improving programmer productivity is simply a matter of using the right computer language has become obsolete; there is now much more emphasis on a variety of programming methodologies and computerized programming support tools. The "Cornell program synthesizer," widely used in basic university computer-science courses, provides a superb demonstration of some current direction in computer science.[20]

While the ultimate aim of the computer science research is to make computer programming far easier than it now is, some of the most promising developments are currently very difficult to work with. To build by hand a computer program that is entirely reliable presently involves an incredibly precise and intricate logic that few programmers have the patience or aptitude to carry through. Much of this logic will have to be incorporated into computerized programming aids where it can be hidden from the average programmer, just as the intricate details of an automobile engine are invisible to the driver of the automobile. Unfortunately, the present state of the art of program development is still too experimental and ill understood to allow rapid development of the fully automated programming systems that are needed.

The computer scientists' effort needs to be supplemented by an equal effort by computer *users*, who will apply their most promising ideas. The users must try these ideas out on problems that exhibit the complexity that is going to be important in the future. These users must find out, through practical experience, whether the complexity barrier can be broken with today's concepts or whether new ones are required. This effort requires users who will fully master the computer science underlying the ideas they apply. The universities have the bulk of the users with the aptitude and motivation to do this. However, these users require adequate facilities. A VAX, even if restricted to a single user, is not powerful enough for many of the applications that will challenge current complexity limits. A Cray-1, or preferably a much more powerful system, is more appropriate, but only if it is shared informally among a small user community. A Floating Point Systems Array Processor is perhaps the least expensive system with adequate power to create the kind of programming problems that need to be studied. Furthermore, the users who are applying computer science ideas need maximum support in other ways. They must be supplied with enormous memory capabilities and with the most advanced graphics terminals available. They must have full access to program development systems that support all the UNIX and LISP-based software being developed by computer scientists and others. (UNIX is a trademark of Western Electirc, Inc.)** They need to be connected to a highly reliable international communications network so that they can instantly exchange ideas and software with either users or computer scientists with similar interests. They must have access to both student and professional help.

These support facilities must be instantly available to junior faculty members, students, or computing support staff members who have good ideas. It should not be necessary to spend two years convincing the senior faculty that one has a good idea and then waiting another two years for a proposal by the senior faculty to be funded.

No university presently has these kind of computing facilities. *No university presently comes close.*

One fallacy that must be avoided is the idea that the hardest problems in computer science can be treated only by computer scientists. For the brightest young people now trying to become involved in computing, labels are meaningless. They may be called physicists, applied mathematicians, psychologists, musicians, or something else. Their actual expertise is governed only by how they choose to spend their time. If their interests require that they learn about modern computer science, they will do so, at an astonishing rate. It makes no sense to exclude these people from working on computer science problems just because their official labels do not read "computer scientist."

In fact, the very great difficulty of the software development problem is due to its not being a problem purely within computer science. The problem of achieving a meeting of the minds between a person and a computer separated by a graphics screen involves aesthetics and psychology just as much as it involves the logic of computer science; the specific application the person has in mind is equally important. The advancement of computer technology is a job for entire universities, not just computer science departments, although the role of computer science is crucial and not adequately appreciated.

Many of the university test and demonstration projects do not involve these deep programming problems. Instead they involve more routine use of new computing systems for teaching and research, using whatever software is currently available. The users of these systems do not build advanced software; instead they discover and report back to the manufacturer any problems that occur within the software and hardware they are supplied with. This is a routine but economically important function that universities can perform cheaper and more effectively than any other segment of the economy and with less disruption to other activities. Both highly advanced software development and the routine testing of new products now have a commercial importance measured in billions of dollars.

E. FINANCING THE UNIVERSITY MARKET

Critical to the multistage design effort I propose is adequate financing of the university market. If the university market is to push innovation in large-scale computing adequately, hundreds of university research groups must be able to purchase multi-million-dollar computing systems in early stages of their development. There must be adequate support for ancillary equipment, for the junior faculty, graduate students, undergraduates, and computer support staff, all of whom would work with these systems, and for software purchases. Support is required for additional floor space and for library support services. The present means of university financing are hopelessly inadequate for this purpose.

It is clear that the technological functions of universities in the modern U.S. economy are all tied to computing technology in one way or another. The very great concern about trained manpower is almost entirely due to the computer; computing technology enhances the value of highly trained personnel and generates a greatly increased demand for them. Every time a new computer is bought, a new source of demand for trained manpower and new technology is created—new demands for programmers, maintenance personnel, and ideas for computer applications, along with a demand for computer technology development to produce future upgrade capabilities for the computer itself. All these demands place new strains on the U.S. university system. To keep up with these demands, as well as to support the university market, it is imperative that support for universities keep pace with the growth of computer technology.

The financing needs of universities could be met, fairly easily, by means of a use tax, namely a retail sales tax on all computing equipment sold in the United States, regardless of

**UNIX is the portable operating system developed at Bell Laboratories. It is widely used in computer science departments. It should become the default operating system for entire university operations. LISP is the language used in the artificial intelligence community.[21]

origin. This use tax would distribute the costs of university support on a roughly equitable basis among the chief economic beneficiaries of the universities' functions. The use tax would define a natural and reasonable scale of university support, determined by their principal economic functions. It would enable university support to grow faster than inflation as long as the computing industry itself grows faster than inflation; this will be a critical requirement once the present recession is past. It would provide new incentives to the universities themselves to maximize their effectiveness in serving the high technology economy.

Tying universities to computing does not mean a narrowing of their normal scope. Universities must learn to use computing technology in support of all phases and fields of their activity—sciences, arts, humanities, sports, administration, and so on. The new support plan, while emphasizing high technology, must enhance all aspects of university life.

No one likes a new tax. However, in the case of the proposed computer tax I think it would be exceptionally difficult to prove that anyone would be harmed even by a very stiff tax. The computer manufacturers would all be treated equably by the tax with respect to both domestic and foreign competition, and would benefit enormously by the increase in trained personnel that would swell their potential customer base. Large commercial users of computing would benefit from more rapid innovation in computing technology and an increased trained manpower pool. In addition, the costs of computing equipment are so volatile that the impact of a computing tax phased in over several years would be difficult to identify.

REFERENCES

Note: References 14–20 published by Floating Point Systems, Inc., P.O. Box 23489, Portland, OR 97223.

1. Wilson, K. "Problems in Physics With Many Scales of Length." *Scientific American,* 241 (Aug. 1979), p. 158.
2. *IBM Journal of Research and Development,* 24 (1980), pp. 107–252.
3. *Electronics,* 54 (1982), p. 41.
4. *Electronics,* 55 (1982), p. 161.
5. Worlton, J. "Supercomputers." *Computerworld,* 15 (1900), p. 82.
6. "Highly Parallel Computing." *Computer,* 15 (1982), entire issue.
7. Batcher, Kenneth E. "Design of a Massively Parallel Processor." *IEEE Transactions on Computers,* C-29 (1980), pp. 836–840.
8. Colton, Bruce. "The Advanced Flexible Processors, Array Architecture." In Peter Lykos and Isaiah Shavitt (eds.), *Supercomputers in Chemistry.* Washington, D.C.: American Chemical Society, 1981, pp. 245–268.
9. Gottlieb, A., and J. T. Schwartz. "Network and Algorithms for Very Large Scale Computation." *Computer,* 15 (1982), pp. 27–36.
10. "Interconnection Networks." *Computer,* 14 (1980), entire issue.
11. Bernhard, R. "Giants in Small Packages." *IEEE Spectrum,* 19 (1982), pp. 39–45.
12. Charlesworth, Alan E. "An Approach to Scientific Array Processing: The Architectural Design of the AP-120B/FPS-164 Family." *Computer,* 14 (1980), pp. 18–27.
13. Wilson, Kenneth G. "Experiences with a Floating Point Systems Array Processor." In G. Rodriguez (ed.), *Parallel Computations.* New York: Academic Press, 1982.
14. Chester, G., R. Gann, R. Gallagher, and A. Grimison. "Computer Simulations of the Melting and Freezing of Simple Systems Using and Array Processor." *Proceedings of the Floating Point Systems 1978 Users Group Meeting: Record,* 1978, pp. 47–58.
15. Bergmark, Donna. "The Design of an AP Fortran Compiler." *Proceedings of the Floating Point Systems 1978 Users Group Meeting: Record,* 1978, pp. 59–72.
16. Giambrone, N., and L. Chace. "AP-190L and IBM 370/168: Software Design and Development Support Scheduling and Control." *Proceedings of the Floating Point Systems 1979 Users Group Meeting: Record,* 1979, pp. 72–89.
17. Bergmark, Donna, and Andrew Hanushevsky. "Document Retrieval: A Novel Application for the AP." *Proceedings of the Floating Point Systems 1980 Users Group Meeting: Record,* 1980, no pagination.
18. Schwartz, Ben. "A Dynamic Segment Loader for the AP." *Proceedings of the Floating Point Systems 1980 Users Group Meeting: Record,* 1980, no pagination.
19. Giambrone, Nicholas. "A Monte Carlo Optimizer for the FPS AP-120B/190L." *Proceedings of the 1981 Array Conference* (FPS), pp. 97–107.
20. Jacobs, Dean, Jan Prins, and Kenneth Wilson, "Monte Carlo Techniques in Code Optimization." *Proceedings of the 1982 Array Conference* (FPS), pp. 44–53.
21. Teitelbaum, T., and T. Reps. "The Cornell Program Synthesizer: A Syntax-Directed Programming Environment." *Communications of the ACM,* 24 (1981), pp. 563–573.
22. *Computer,* 14 (1981), no. 4 (entire issue).

Dynamic RAM architectures for graphics applications

by DOUGLAS L. FINKE
Intel Corporation
Aloha, Oregon

ABSTRACT

This paper explores the requirements for future graphics memories and analyzes the requirements for two of the biggest problems in graphics terminal design, raster display bandwidth and random-bit update performance. To solve these problems a new access mode for dynamic RAMs called Ripplemode™ will be proposed. This mode directly solves the raster display bandwidth problem by providing high-speed access to a serial bit stream. Under certain circumstances this mode also improves the performance of random-bit updates.

INTRODUCTION

For the past decade dynamic RAM has been the memory technology of choice for implementing the bit-mapped approach to graphics memories. The characteristics that make dynamic RAM so desirable include lowest cost per bit, reasonable speed, high packing density, and low power. As the cost performance of dynamic RAM memory has steadily improved, system designers have been quick to take advantage of these improvements by increasing the capabilities of their displays. Whereas a common display size in past years may have been 384 × 512 pixels with 1 bit per pixel, displays of beyond 1024 × 1280 pixels with more than 8 bits per pixel will be common in the future.

As the demands for improved performance continue to increase, it is becoming apparent that the optimum chip architecture for graphics applications is diverging from that historically used for dynamic RAMs. The trend to higher-density chips, faster clock cycles, color displays, and animation is causing memory producers to seek out alternative RAM architectures to provide solutions to these problems in a cost-effective manner.

This paper explores the requirements for future graphics memories and analyzes the requirements for two of the biggest problems in graphics terminal design, raster display bandwidth and random-bit update performance. To solve these problems, a new access mode for dynamic RAMs called Ripplemode will be proposed. This mode directly solves the raster display bandwidth by providing high-speed access to a serial bit stream. Under certain circumstances this mode can also improve the performance of random-bit updates.

RASTER DISPLAY BANDWIDTH REQUIREMENTS

A raster scan display is composed of three basic time blocks (shown diagramatically in Figure 1). Area 1 represents the time block used to display the image pixels onto the screen. Its magnitude is simply the product of the number of lines per screen, the number of pixels per line, and the average time per pixel. Area 2 represents the time required for horizontal blanking, during which horizontal retrace occurs. Area 3 represents the vertical blanking time, during which vertical retrace occurs. During Periods 2 and 3 the pixel memory is not being used for the raster-scanning function. The sum of Periods 1, 2, and 3 represents 1 display frame time.

Table I shows an analysis of the actual pixel time for four typical graphics implementations. It assumes a 60-hz noninterlaced display with aspect ratios of approximately 3:4. The horizontal and vertical blanking times are nominal values chosen on the basis of monitor performance specifications.

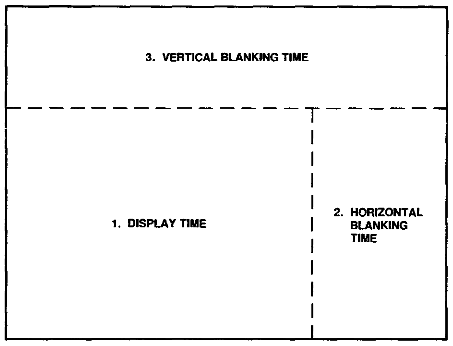


Figure 1—Frame time partitioning

Typical graphics systems use multiple memory chips per system, for two separate reasons. First, with today's density chips, several memory devices are required just to hold the required capacity. But a second factor is that typical implementations are required to parallel several devices in order to achieve the required data rates. An upcoming problem with high-density devices is the possible mismatch between these requirements. As Table II shows, bits can be wasted with standard dynamic RAMs because the number of devices needed to achieve the required bandwidth can be higher than the number of devices needed to supply the bits. Alternative architectures are necessary if one wants to use 64K × 1 and 256K × 1 dynamic RAMs without wasting bits.

One important characteristic of raster-scanning accesses is their serial nature. Dynamic RAM architectures that provide fast access to serial data streams can take advantage of this characteristic to help meet the raster-scanning requirements.

One of the oldest forms of such serial access is known as page mode. Inside a dynamic RAM the addresses are divided into rows and columns. A 64-K dynamic RAM, for instance, has 16 address bits, of which 8 are for the rows and the other 8 for the columns. Because of the internal architecture of the part, access to a new column address within the same row can be made somewhat faster. In the 2164A,[1] for example, a new

Table I—Full frame scan specifications

Display Pixels	Lines	Pixels Per Line	Vertical Blanking Time	Horizontal Blanking Time	Display Time	Total Blanking Time	Pixel Time
196 K	384	512	900 μs	7.0 μs	13.08 ms	3.58 ms	66.5 ns
328 K	512	640	800 μs	6.5 μs	12.54 ms	4.12 ms	38.3 ns
786 K	768	1024	700 μs	6.0 μs	11.36 ms	5.30 ms	14.4 ns
1310 K	1024	1280	650 μs	5.4 μs	10.49 ms	6.17 ms	8.0 ns

Table II—Devices required for bandwidth versus memory space

Display Pixels	Minimum Devices in Parallel	Minimum Devices to Supply Bit Capacity			% Bit Wasted		
		16K	64K	256K	16K	64K	256K
196K	5	12	3	1	0	40	80
328K	8	20	5	2	0	38	75
786K	21	48	12	3	0	43	86
1310K	38	80	20	5	0	47	87

This table was calculated by assuming a 300-ns cycle time for single bit reads, the pixel time requirements from Table I, and a single bit/pixel.

row address can be accessed in 150 ns and cycled in 260 ns, and a new column address can be accessed in page mode in 85 ns and cycled in 125 ns. This provides an improvement of about a factor of 2 for raster scanning. A further improvement can be made to page mode by extending the time length over which it can be used from 10 μsec to a much longer 75 μsec. The significance of this improvement, known as Extended Page Mode, lies in that fact that the maximum time length between successive horizontal retraces is always less than 75 μs and that therefore page mode can be in use for one entire horizontal sweep. Another important advantage is that it allows one to page out the entire 256 bits in a row. With a limit of 10 μs for page mode only 80 bits can be paged out before the device must be put through a precharge cycle. Extended page mode on the 2164A can make low-density displays, such as 256 × 256 pixels, very easy to implement. Each horizontal line on the display can be mapped to a row within the chip address space, and extended page mode can be used to access the entire line at the required data rate. Only one memory chip is required to implement the entire memory (assuming 1 bit/pixel) with no interleaving, parallel to serial converters, or other complications.

A second approach to improving the memory bandwidth is simply to make the memories wider.[2] By organizing the memories as either ×4 or ×8, the bandwidth is obviously improved by a factor of either 4 or 8. In Table II, for example, one could show that a memory organized either as 8K × 8 or 16K × 4 could be used without wasting any bits, whereas its 64K × 1 counterpart wasted about 40%, as shown. There are disadvantages to this approach, however. One needs to convert the parallel outputs to a serial stream, adding extra cost and complexity to the design. In addition, these organizations all use common I/O, thus raising the possibility of bus contention, complicating data path bussing, and adding extra loading and capacitance to slow down the data transfer rate. In addition, these parts are typically packaged in larger packages than ×1 parts, increasing the end system's physical size. Finally, problems can occur when one wants to write only a subset of the 4 or 8 bits in the chip. To accomplish this, a slower and more complicated read-modify-write cycle must be used instead of the normal write.

A third approach to providing faster sequential access is called Nibblemode.[3] This approach attempts to provide the advantages of a ×4 architecture without suffering from any of the problems. In a 64K chip, for example, a Nibblemode device is internally organized as 16K × 4. However, instead of all 4 bits' being sent directly to the output pins, they are latched into a shift register. On successive cycles of the CAS clock a new bit appears on the output pin, and access to the 4 serial bits can occur at a rate of 40 to 60 ns. This method provides high peak bandwidth for short bursts while avoiding all the problems of common I/O and parallel outputs. The main disadvantage of this approach is that it is only 4 bits. Although the peak bandwidth is high, the average bandwidth is less. That is because this method still does not eliminate either the large access times for the first bit or the precharge time for the next cycle after the fourth bit has been accessed. Looking at it in another way, this method provides a slow access to the first bit in a series and then three fast accesses to the second, third, and fourth bits.

After examining the alternatives, Intel has committed its future line of dynamic RAMs to a new form of access mode called Ripplemode. This new mode is functionally compatible with extended page mode but provides greater speed and bandwidth than any of the modes mentioned above with none of the associated problems. Ripplemode is made possible by the use of a combination of HMOS-III-level technology and innovative circuit design techniques.

Like extended page mode, Ripplemode provides quicker access to any bit within the same row at a fast rate by sending out a new column address and clocking the CAS clock to latch it in. Compared to extended page mode, however, Ripplemode provides sequential cycle times at the chip level of only 40 ns, compared to the 125 ns of the 2164A. Ripplemode further improves performance at the system level by incorporating a lookahead function in the column address buffers. Because this circuitry is implemented by using flowthrough latches instead of edge-triggered latches, an access can be initiated even before the CAS clock has been brought low. So instead of only a 2× speedup, Ripplemode provides a 6× speedup over the standard random-access mode. The advantages of Ripplemode include the fact that it provides access to 256 or 512 bits instead of only 4, it does not depend on common I/O, and it provides more flexibility in the access pattern. Bits can be read forward, backward, or even pseudorandomly without any degradation in access time as long as the bits are all within the same row. In addition, Ripplemode is fully upward-compatible with page and extended page mode. Any design that used these older modes can plug in Ripplemode parts without any modification.

To illustrate the power of Ripplemode, a 1-chip implementation of the 384 × 512 display can be accomplished just by using a 256-K × 1 dynamic RAM with Ripplemode. Again, this implementation would not require any interleaving, parallel-to-serial converters, or other complications. It would provide not only the best performance levels, but also the simplest system implementation.

Although Ripplemode does not require usage of common I/O to achieve its bandwidth, there is nothing to prevent the combining of Ripplemode with a ×4 or ×8 organization to improve the levels of bandwidth further. A 64K × 4 with Ripplemode, for example, can provide data rates of up to 100 MHz for blocks of 1024 bits for applications that require the ultimate in dynamic RAM bandwidth.

PIXEL UPDATE PERFORMANCE

Once the problem of obtaining adequate bandwidth for raster displays has been solved, a second one can appear. Not only does one need to display the contents of the memory on the screen; one must also be able to update the memory in order to change the display. Unlike the bandwidth requirements for raster display, which can be precisely quantified (as shown above), the bandwidth requirements for pixel update vary greatly with each application.

As is shown in the following discussion, Ripplemode is also helpful for solving the pixel update problem. One of the major advantages is that it alleviates the need to use a ×4 or ×8 organization in the memory. Because a pixel generator typically only writes 1 pixel at a time, updates using a ×4 or ×8 common I/O architecture must be accomplished by using a read-modify-write cycle. Besides complicating the design, requiring read-modify-writes slows down the write performance of the memory considerably. Whereas a standard write cycle can be easily accomplished within a system in 300 ns, the system cycle time for commonly available parts is closer to 400 ns.

An application with only moderate update requirements could be one that displays a series of static screens without any constraints on the total display time. Another possibility could be that of a radar display. Although the screens of such a display are constantly changing, they never make complete scene changes from frame to frame. A small segment of the screen may change within 1 frame time, but these changes usually represent just a small fraction of the total pixels. As shown in Table III, these applications can afford simply to accomplish updates in standard random-access mode during the retrace periods and still meet their performance requirements.

Table III represents the performance one can obtain by using standard random-access writes during the retrace periods to write the screen. It assumes a 300-ns system cycle time for the write, and enough refresh cycles to satisfy the refresh requirements of a 256-cycle/4-msec or a 128-cycle/2-msec DRAM. It also assumes a frame time of 1/60 sec and flicker-free operation.

What Table III shows is that even if the update cycles are limited to the retrace time within a frame, a full screen can be completely written within 0.3 to 1.5 sec. Put another way, this scheme for accomplishing updates can accommodate a pixel generator that produces a new pixel every 862–1628 ns.

In many systems the bottleneck is not actually within the memory system but instead in the pixel generator itself. Although performing pixel updates during retrace periods provides adequate average bandwidth for many applications, a mismatch can occur between the peak bandwidth demands the pixel generator demands and the peak bandwidth the memory is able to supply. If the processor needs to wait until the memory is available, the system throughput can be slowed down considerably. To solve this problem, a pixel write buffer can be placed between the pixel generator and the graphics memory. By using three 2 K × 8's to queue an update's address and data, one can obtain near-full-speed performance from both the memory and the processor.

A slightly more stringent application could be one with a requirement that full screen clears or any other block-type function be accomplished quickly. As shown below, the requirements of these applications can be met by taking advantage of Ripplemode to accomplish the full screen clears and then using standard random accesses during retrace to accomplish the rest.

Since clearing the screen is one of the most common graphics functions, a considerable improvement in this function can be made with very little extra hardware. Table IV shows the time it would take to accomplish this function with Ripplemode. It assumes a 75-ns system cycle time for Ripplemode and the use of Ripplemode functions only during the vertical blanking time, since the horizontal blanking time is not long enough to allow for rippling through an entire row.

Finally, there are applications that require extremely large amounts of update bandwidth. Characteristics of these applications may include the use of animation, very rapid scene changes, or just a large amount of movement from frame to frame. For these applications the use of a double-buffered memory is recommended.

Although it may seem to be a waste to use twice the number of bits needed for display purposes, the added cost and density may actually turn out to be minimal, especially for systems of this performance level. With 256-K technology, the 1310-K pixel display will be implemented with only 5 chips per memory plane. Even with as many as 4 bits per pixel, a fully double-buffered 1024 × 1280 × 4 memory system will fit onto 1 card.

Double buffering works with 2 equal-sized frame buffers. While the primary display buffer is used to serve the raster-scanning requirements, the pixel generator has full access to the secondary buffer to make updates. At the appropriate time, the system will switch buffers so that the updated buffer is displayed on the screen and the old display buffer is now used for updating. In some cases it may be necessary to copy the new display buffer into the update buffer before the pixel generator can use the update buffer. This can be accomplished in 1 frame time by using Ripplemode. While the display buffer is being read in Ripplemode for raster-scan-

Table III—Full-screen write time

Display	Blanking Time	Maximum Random Cycles During Retrace time	Refresh Cycles Per Frame	Update Cycles Per Frame	Full-Screen Write Time
196 K	3.58 ms	11,933	1280	10,653	0.32 sec
328 K	4.12 ms	13,733	1280	12,453	0.45 sec
786 K	5.30 ms	17,666	1280	16,386	0.80 sec
1310 K	6.17 ms	20,566	1280	19,286	1.13 sec

Table IV—Screen clear time using Ripplemode

Display	Vertical Blanking Time	64 K × 1 or 64 K × 4	256 K × 1
196 K	900 μs	0.10 sec	0.38 sec
328 K	800 μs	0.12 sec	0.43 sec
786 K	700 μs	0.13 sec	0.50 sec
1310 K	650 μs	0.13 sec	0.53 sec

Table V—Full-screen write times using double buffering

Display	Random Cycles Per Frame	Refresh Cycles	Update Cycles	Full-Screen Write Time
196 K	55,555	1280	54,275	0.07 sec
328 K	55,555	1280	54,275	0.12 sec
786 K	55,555	1280	54,275	0.25 sec
1310 K	55,555	1280	54,275	0.42 sec

ning purposes, the same data can also be fed into the update buffer and written with Ripplemode. As Table V shows, the performance improvement with this technique can be considerable.

DESIGN EXAMPLES

Using the concepts explained above, some design examples with Ripplemode RAMs are briefly described in this section. Although these examples are shown using only 1 bit/pixel system for simplicity's sake, extending them to multiple bit/pixel systems is very straightforward.

The first system, a moderate-performance one, is shown in Figure 2. Although Figure 2 is shown using a single 256-K × 1 RAM, it could just as easily be implemented with three 64 K × 1's. As was shown earlier, the 196-K system requires a pixel cycle time of 66.5 ns. This requirement can be met with a single chip when Ripplemode is used. As a result, a shift register is not required to accomplish the parallel-to-serial conversion. Since the length of a line is 512 pixels, it exactly matches one row in the 256 K. When using 64 Ks, 2 rows in different chips are required, since 64 K rows are only 256 bits long. The memory-refresh requirements can be met by performing 3 refresh cycles during the horizontal retrace period at the end of each line. This system would allow memory updates to occur only when the memory is not being used for the raster display or refreshes.

The second example in Figure 3 is a medium-performance system with enough memory for a 328- to 512-K pixel display. It also includes provisions for Ripplemode blanking and a 2-K pixel update buffer to improve write performance. In this system the address and data for pixel updates are loaded into a FIFO memory, allowing the processor to generate new pixels as long as the FIFO is not full. The display controller in turn reads the FIFO and updates the display memory during

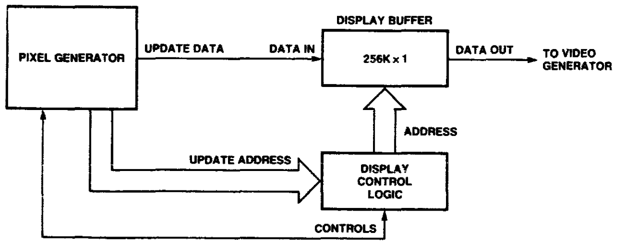


Figure 2—Moderate-performance 196K–256K display system

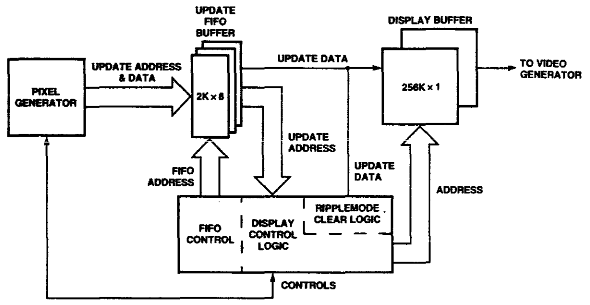


Figure 3—Medium-performance 328K–512K with fast clear and update FIFO

the times the display memory is not busy. The display memory can also use Ripplemode to improve performance for block fill and clear operations.

The final example, Figure 4, is an ultra-high-performance system. It contains two 1310-K buffers to provide support for simultaneous raster display and updates. Because of the extremely high data rate required (8 ns/pixel), 64 K × 4s are used to read out a total of 20 bits in parallel; and these are converted to a serial stream by a 20-bit shift register. This application only requires 1 or 2 refreshes per line. Control signals can be sent from the pixel generator to the display control logic when it is time to switch buffers.

SUMMARY

One thing is becoming clear as the capabilities of dynamic RAM memories continue to increase. With today's technology it is becoming technically feasible and more desirable to optimize RAM chip architecture for varying applications. Architectures for, say, mainframe memories and graphics displays will be considerably different in the future.

This paper has shown the first steps in optimizing memory architectures for applications in graphics. Even more innovative structures are being investigated, and in the future architectures that make possible the implementation of an entire high-performance bit map memory within a single chip will be possible.

ACKNOWLEDGMENTS

The author is indebted to Charles Guy for his considerable help in analyzing graphics memory requirements and to William Righter for suggesting improvements to preceding drafts of this paper.

REFERENCES

1. 2164A Data Sheet, Intel Corporation, 1982.
2. Cole, Peyton M., David W. Gulley, and Lionel White. "Wide-Word Memory Chips Spur New Microprocessor Applications," *Electronic Design*, November 26, 1981, pp. 231–238.
3. Eaton, S. Sheffield, David Wooten, William Slemmer, and James Brady. "A 100 ns. 64K Dynamic RAM using Redundancy Techniques." *1981 IEEE Solid-State Circuits Conference Digest of Technical Papers.* Coral Gables, Fla.: IEEE, 1981. pp. 84–85.

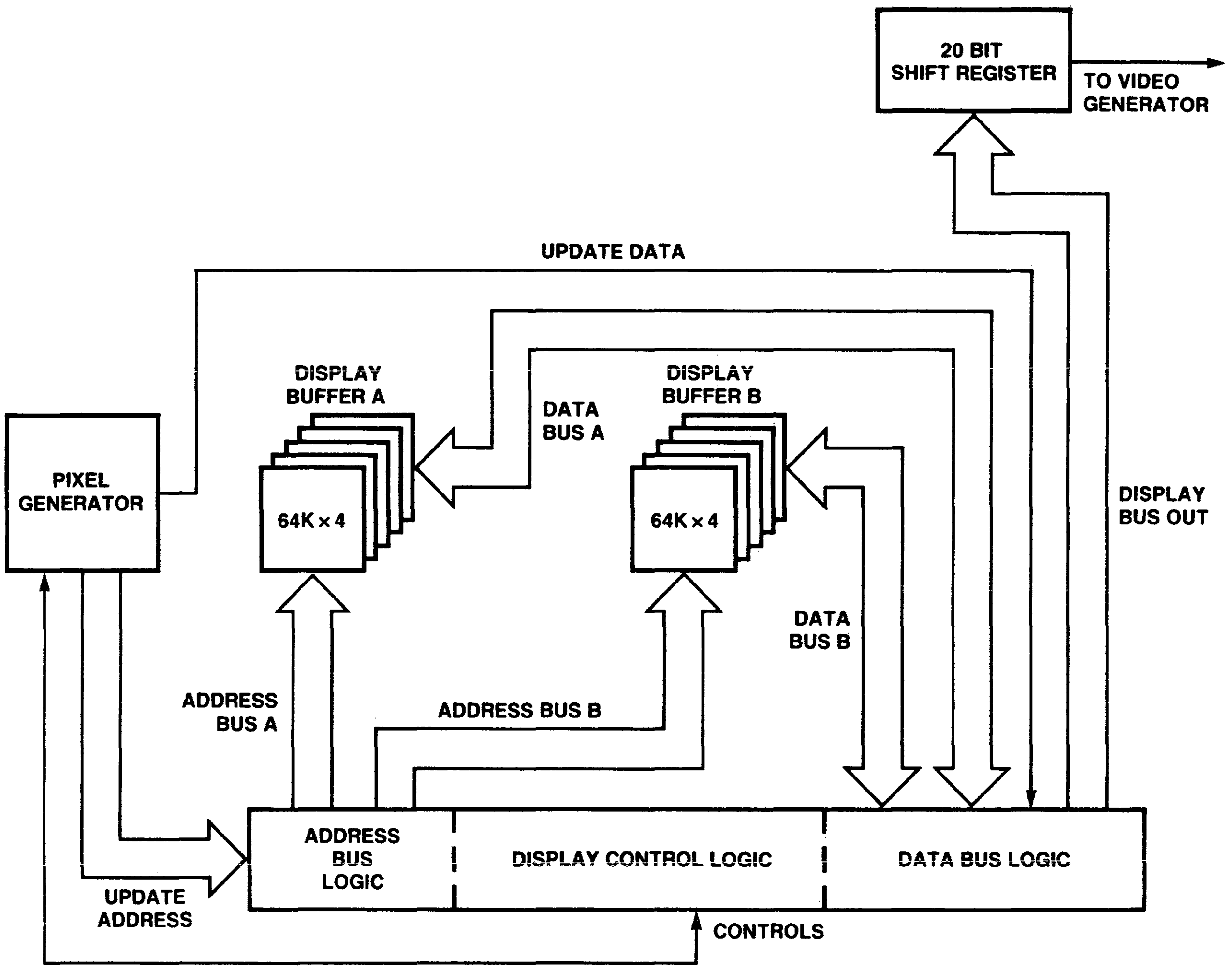


Figure 4—High-performance, double-buffered 1310K display

The iRAM—an innovative approach to microprocessor memory solutions

by JOHN J. FALLIN
Intel Corporation
Aloha, Oregon

ABSTRACT

When designing the local read/write memory for a microprocessor system, several specific needs should be addressed. These needs include flexibility, ease of use, and low cost. The iRAM, a new product, effectively addresses all of these needs. The iRAM is a complete memory system that combines a high-density 8K × 8 DRAM array with complete memory and refresh control on a single chip. The iRAM combines the advantages of both the DRAM and SRAM, making it the ideal choice for microprocessor memory. The iRAM's concept and features will be discussed as well as its use in local memory systems for both 8-bit and 16-bit microprocessors.

MICROPROCESSOR MEMORY NEEDS

With the advent of microprocessors, more system designers are involved in the design of microprocessor local memories. Analysis of the memory system highlights several important memory characteristics: ease of use, low cost, and flexibility. In the past, the designer has had to decide on the relative importance of each of these characteristics and then make a choice between DRAMs and SRAMs, neither of which satisfies all of the requirements in all cases.

Ease of Use

To understand how a new memory, the iRAM, meets all of these system criteria, it is necessary to examine why these three characteristics are so important. Ease of use simplifies the interface between the memory and the microprocessor. This translates into an easier, shorter design cycle while minimizing the component count. The advantages to the manufacturer include quicker time to market, lower cost, and less board-area requirements.

Cost

If ease of use were the only criterion used in selecting a memory component for the microprocessor local bus, then the SRAM would be the obvious choice. The SRAM is the simplest memory to use. The SRAM, however, has a complex storage cell, making it more difficult to manufacture at the higher densities and thus more expensive. Excessive cost limits SRAMs to small memory systems (those typically less than 8K bytes).

Higher density and corresponding lower cost allow DRAMs to be used in larger memories. The refresh requirement of DRAMs, however, makes the system more complex and difficult to design. The cost of the refresh control, however, can be amortized in large systems (those greater than 64K), making DRAMs very cost-effective in this memory segment.

Flexibility

Flexibility is another characteristic that simplifies the design cycle. Rather than design separate systems for different models of the end product or redesign for an upgrade, which only requires more memory, the designer can design around the universal site. This is a JEDEC standard site (Figure 1), which is capable of accommodating other JEDEC standard memory devices, both volatile and nonvolatile RAMs, EPROMs, PROMs, and E^2PROMs.

Many advantages are gained through the use of the universal site. In the early design phase, the mix between ROM, EPROM, and RAM may not be known. Using the universal site can allow the hardware design to be completed at this preliminary stage. This permits software to be debugged on the final product and the RAM/ROM mix to be easily varied later. The universal site also allows for easy upgrades because memory devices from 2K to 16K fit the site.

As mentioned earlier, the SRAM, with its easy-to-use bus structure, fits nicely in small systems (those 8K and smaller). For large systems (those greater than 64K), where the cost of using SRAMs becomes prohibitive, DRAMs are used. Between 8K and 64K a transition region exists in which neither SRAMs or DRAMs are a good fit. In this region low cost, ease of use, and flexibility are all important. In the past, compromises had to be made between these features. Now, because of an innovative new development in RAM technology, these compromises are eliminated. The iRAM, which is the first in a generation of intelligent memories, is a complete memory system on a single silicon chip. The iRAM achieves low cost through the use of a high-density DRAM array. It is easy to use because of its onboard refresh control and is manufactured in a JEDEC compatible 28-pin package, making it very flexible.

A SOLUTION: THE iRAM

The Intel iRAM comes in two different varieties: the 2186 and the 2187. Both iRAMs are 5-volt only DRAM subsystems organized as 8192 8-bit words, but they operate differently. In most microprocessor systems, the asynchronously refreshed iRAM, the 2186, is used. This iRAM incorporates a refresh timer to maintain refresh. To synchronize refresh and access cycles, an onboard arbiter queues cycles and provides a ready handshake signal (RDY) to indicate a pushout in the cycle. (The RDY output is commonly used to request the insertion of WAIT states when required). The 2187 replaces the RDY output with a refresh enable ($\overline{\text{REFEN}}$) input. During 2187 operation, the user strobes $\overline{\text{REFEN}}$ to maintain refresh. The 2187 is ideal for use in systems where WAIT states cannot be tolerated, or in systems where accesses occur in a manner adequate to maintain refresh (such as in some graphics systems).

Internal Structure

A block diagram of the iRAM is shown in Figure 2. For ease of discussion, the iRAM is divided into five basic sections; a 65,536-bit DRAM array, a refresh-request time, a refresh

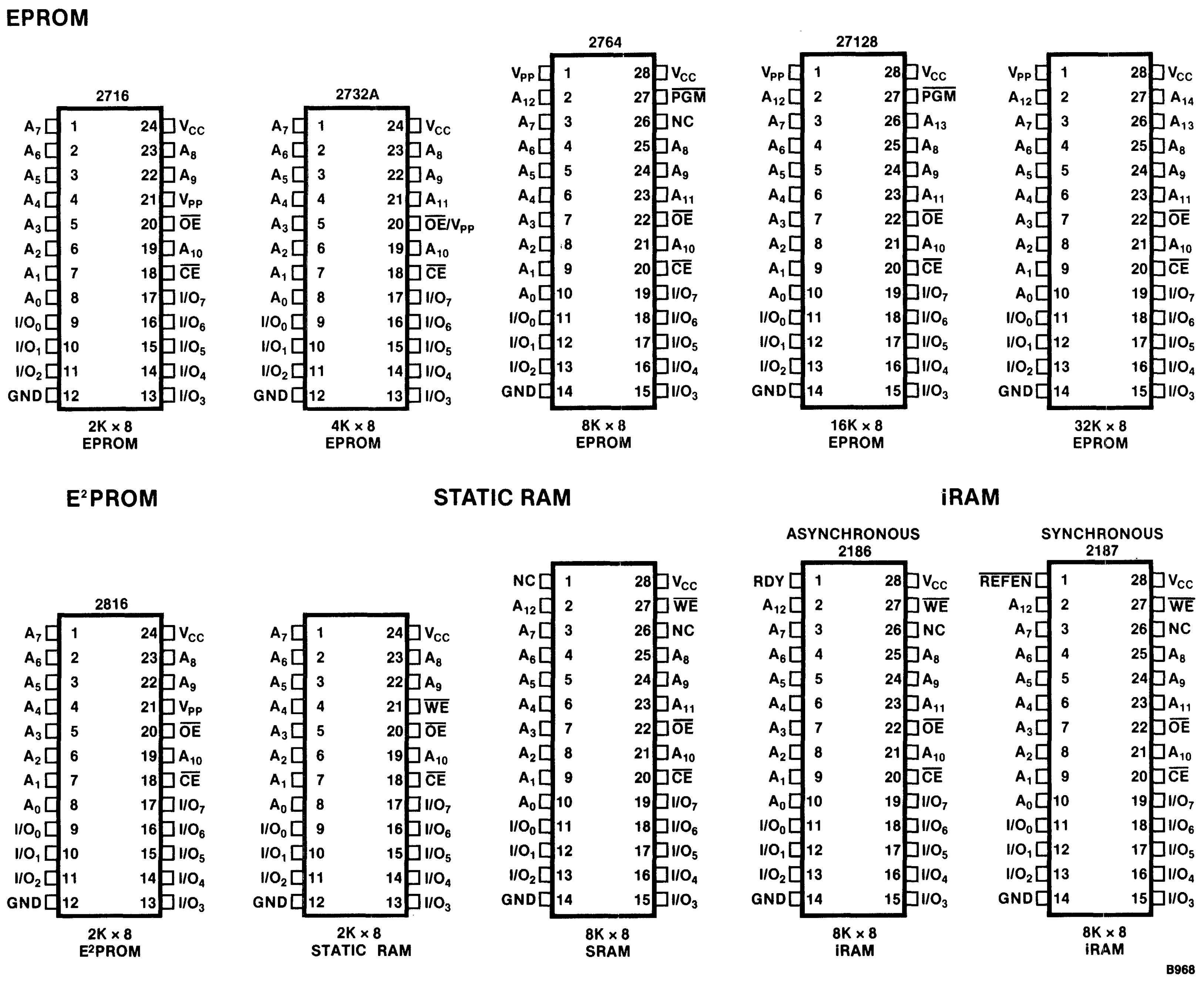


Figure 1—Universal site

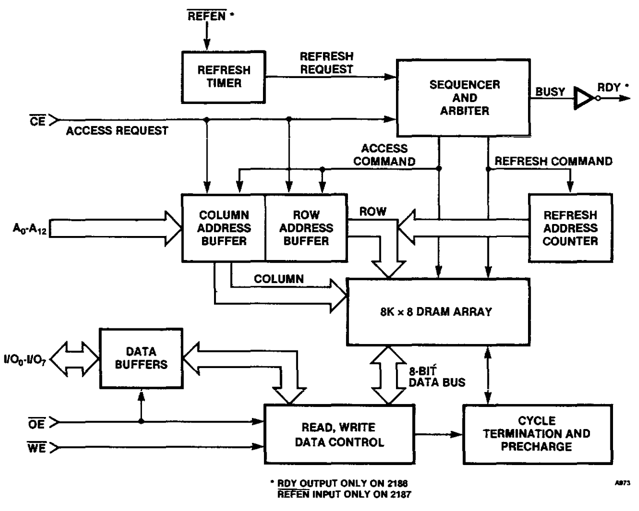


Figure 2—2186/2187 block diagram

row-address counter, a high-speed cycle arbiter, and full control circuitry.

The refresh timer provides refresh requests to the arbiter, which synchronizes the refresh cycles with access cycles. Refresh addresses are generated by the refresh row-address counter, multiplexed with external row addresses and then routed to the DRAM array. This DRAM array is divided into four quadrants, each containing 128 rows and 128 columns. Internal control circuitry synchronizes all internal events. The 2187 differs from the 2186 in that it has no arbiter and the refresh timer is user-controlled, instead of being free running.

iRAM Operation

The pinouts of the 2186 and the 2187 are shown in Figure 1. Note that, except for pin 1, the iRAMs have identical pinouts to the 8K × 8 SRAM and EPROM, providing flexibility for the RAM/ROM mix in the system. The 2186 operates much

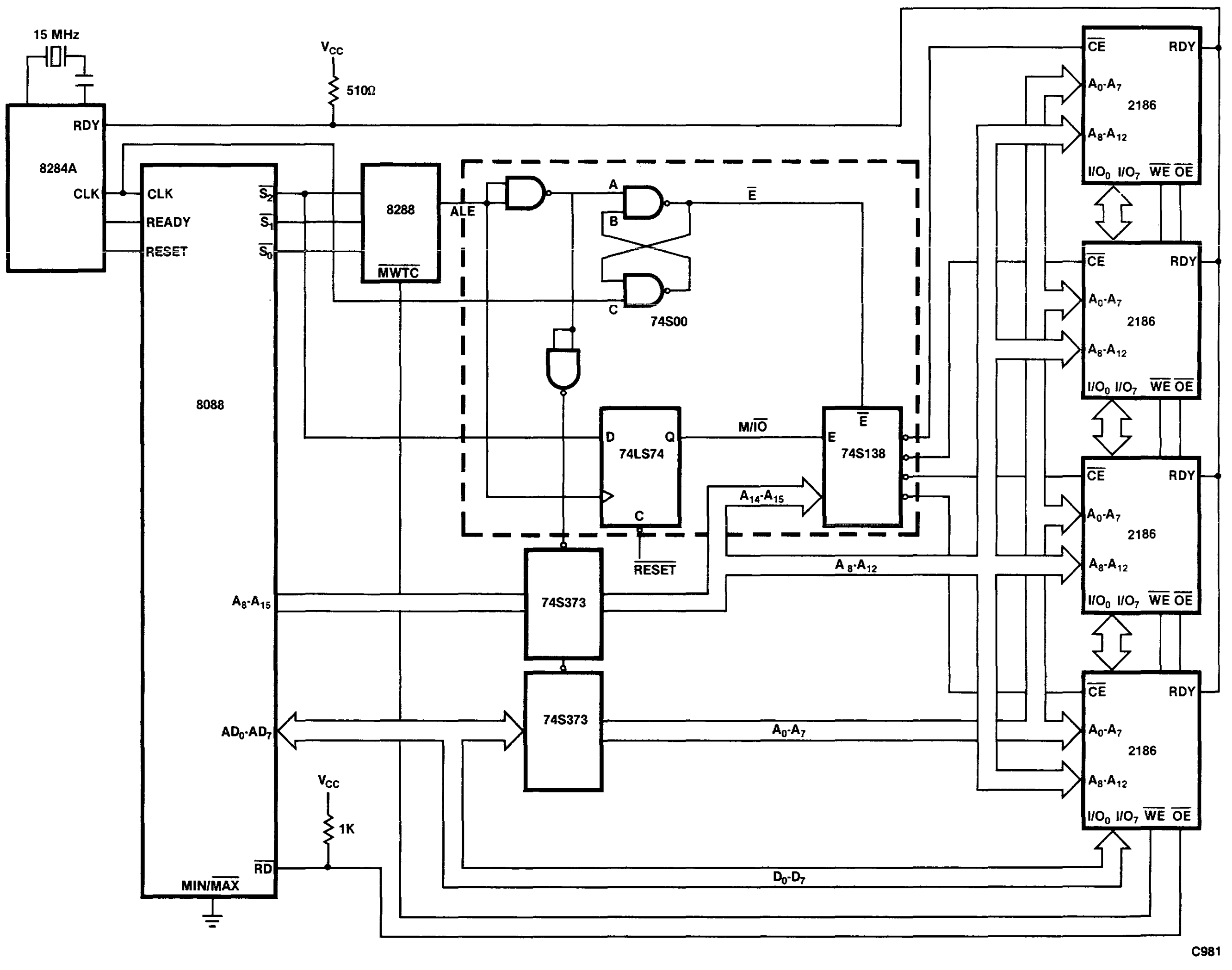


Figure 3—8088/2186 system

like an SRAM during read and write cycles. The only differences involve the $\overline{CE}$ and $\overline{WE}$ inputs and the RDY output (2186) or the $\overline{REFEN}$ input (2187). Because the iRAM is an edge-triggered device, its control inputs must be clean transitioning. Also, because of the dynamic nature of the device, its addresses are latched on the leading (falling) edge of $\overline{LE}$, eliminating latches in some systems. Both of these requirements are very easy to implement at the system level, as will be shown later. Another difference of the iRAM is that during a write cycle, the iRAM latches in data on the leading (falling) edge of $\overline{WE}$, as opposed to the SRAM, which latches data on the trailing edge of $\overline{WE}$.

During operation, when the 2186 is accessed, the RDY output will occasionally be pulled low, indicating a deferred cycle. This RDY output is normally routed back to the microprocessor READY or WAIT input. WAIT states are injected only when required during an access/refresh conflict.

Another type of cycle that exists for the iRAM is the false memory cycle, in which the iRAM receives an active $\overline{CE}$ but then neither $\overline{OE}$ or $\overline{WE}$ comes low. The false memory cycle is somewhat like a $\overline{RAS}$-only refresh in that the row selected by the seven external row addresses is refreshed.

The 2187 operates slightly differently than 2186 in that the user now strobes the $\overline{REFEN}$ input to maintain refresh. The $\overline{REFEN}$ strobe must be timed such that access cycles are not attempted during refresh cycles.

APPLICATION EXAMPLES

In the section that follows, several microprocessor/iRAM interfaces will be discussed to demonstrate the iRAMs flexibility and ease of use. These will include both 8-bit and 16-bit systems, as well as a synchronous 2187 design. Also included is an iRAM/microcontroller system design.

8088/2186

Figure 3 shows the simplicity of system design in an 8088/2186 max mode system that runs at 5 MHz without WAIT states. Operating like a clocked static RAM, the iRAM requires stable addresses on the falling edge of a $\overline{CE}$, which should be transient-free to prevent multiple selection of the iRAM. Only one TTL device is needed to perform this function.

To guarantee a clean transitioning $\overline{CE}$, the $\overline{CE}$ decoder is only enabled when its address inputs are stable. This is accomplished by using a cross-coupled NAND gate to generate a decoder-enable signal. Also used as a decoder enable is status

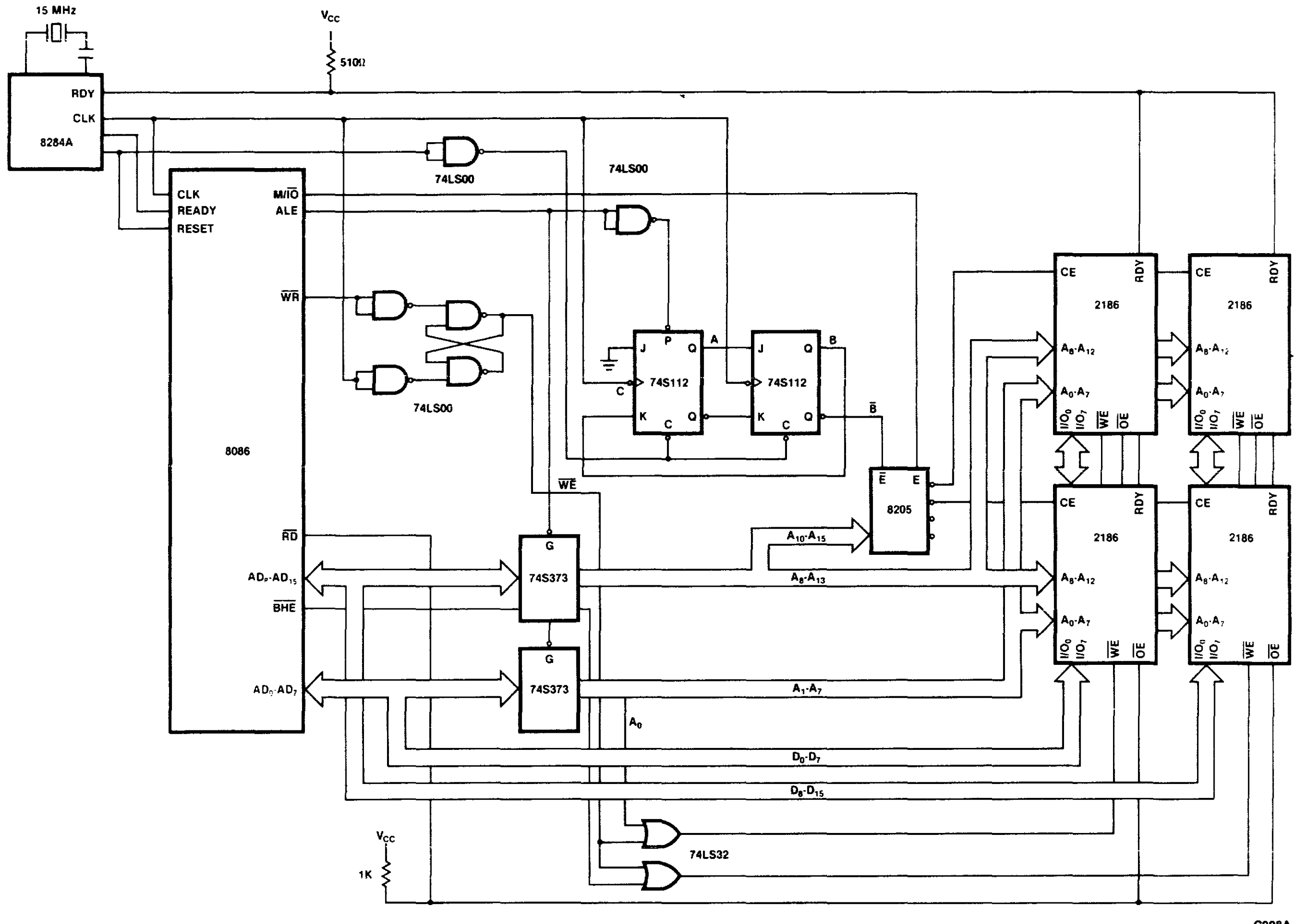


Figure 4—8086/2186 system

bit 2 (M/$\overline{\text{IO}}$). This guarantees that a $\overline{\text{CE}}$ will only occur if the ALE output from the processor is associated with a memory read or write cycle. This eliminates false memory cycles, which have an extended $\overline{\text{CE}}$ high-time requirement not allowed for in this design. Note that the address inputs to the decoder are latched, because addresses can transition between bus cycles even if ALE is not high. Because the decoder is enabled during this period, transitions could propagate through the decoder and cause invalid $\overline{\text{CE}}$s to occur, jeopardizing data integrity.

The $\overline{\text{WE}}$s for the iRAM array are generated directly from the 8288 bus controller. The 8288 output $\overline{\text{MWTC}}$ has a delayed falling edge, which allows for the leading-edge write requirement of the iRAM, while providing compatibility with SRAMs.

8086/2186

The 8086/2186 system shown in Figure 4 is similar to the previously described 8088 system and also runs at 5 MHz with zero WAIT states using a 2186-30. The major difference is in the $\overline{\text{CE}}$ generation circuitry. Because this is a 16-bit system, a 2186 can receive a $\overline{\text{CE}}$ but no $\overline{\text{WE}}$ or $\overline{\text{OE}}$, resulting in a false memory cycle. This condition occurs in the unwritten byte of the 16-bit word during a byte-write operation. The longer $\overline{\text{CE}}$ high-time requirement of the false memory cycle can be accommodated by using the $\overline{\text{CE}}$ generation circuitry shown in Figure 4. The circuitry is basically a 2-bit counter that initiates

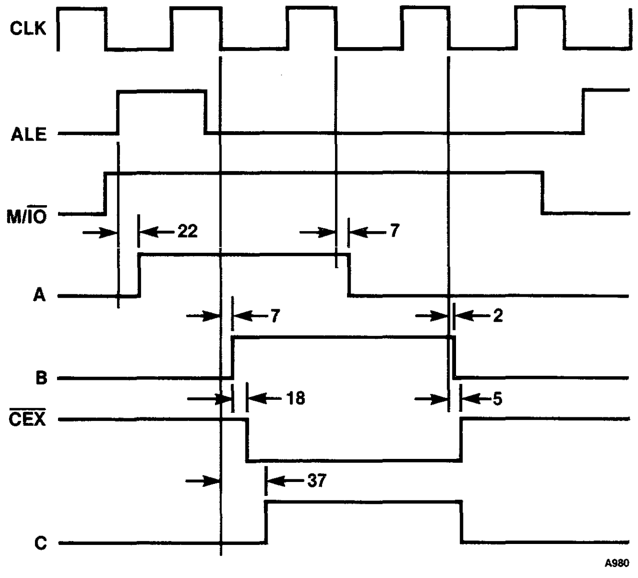


Figure 5—$\overline{\text{CE}}$ circuit timings

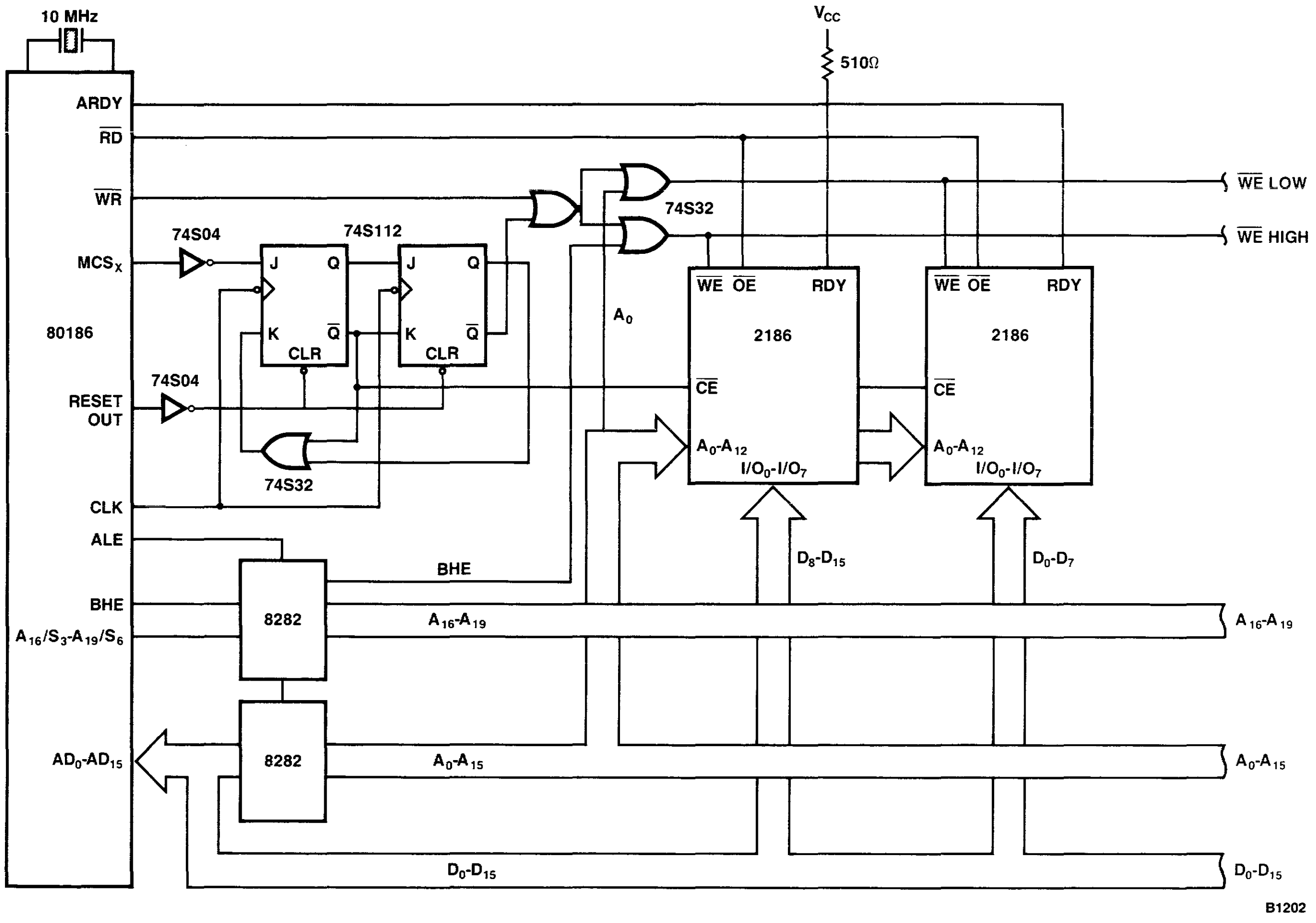


Figure 6—80186/2186 system

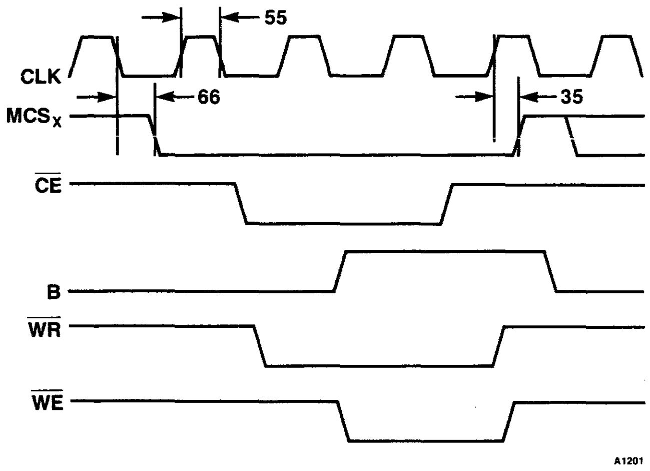


Figure 7—80186/2186 timings

its count on the rising edge of ALE. On the next clock (falling edge) after this, $\overline{CE}$ is activated and remains so for two clocks before returning high. This provides a $\overline{CE}$ high time of approximately two clock periods, or, for a 5 MHz 8086, approximately 400 ns, sufficient to meet the $\overline{CE}$ high-time requirement. Timings for this circuit are shown in Figure 5.

Because the 8086 is configured in the min mode, its $\overline{WR}$ output falls too early in the cycle to satisfy the leading-edge write requirement of the iRAM. To meet this requirement, a cross-coupled NAND gate circuit delays $\overline{WE}$'s falling edge. The $\overline{WE}$ rising edge, however, is not delayed, which allows for SRAM compatibility.

80186/2186

Figure 6 depicts an 80186/2186 interface. The 80186 is a highly integrated device that combines an enhanced 8086

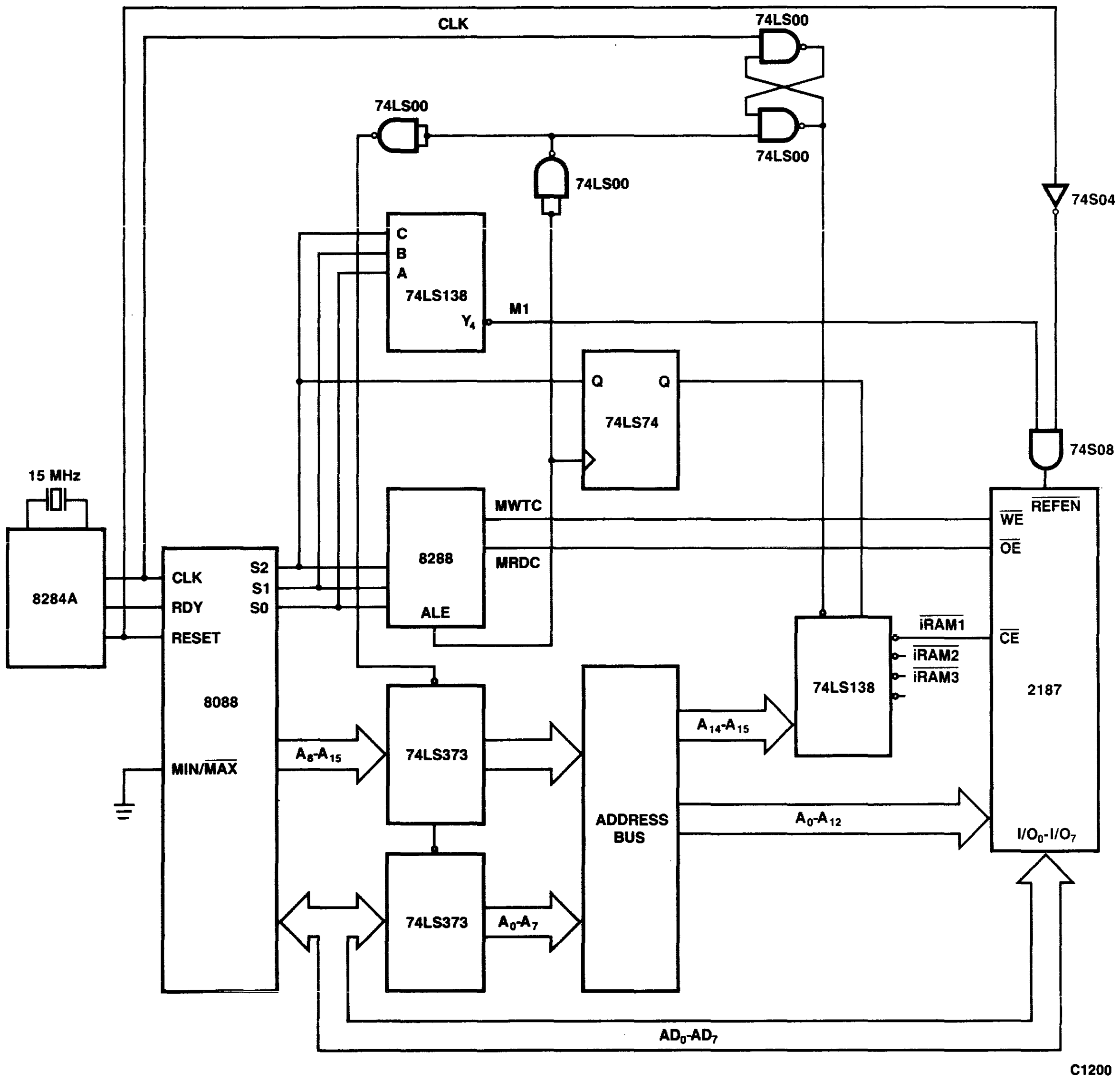


Figure 8—8088/2187 system

CPU with 10 commonly used system components, including a bus driver, clock generator, interrupt controllers, and programmable memory chip selects.

The circuitry used to generate the $\overline{CE}$ is similar to that for the 8086 system in the last example. Again, a 2-bit counter is used, although its count is initiated differently. A programmable memory chip select is used to toggle the first flip-flop from a zero to a one on the first clock. This action causes a $\overline{CE}$ to be generated. On the next clock, the first flip-flop will remain set and the second flip-flop will also be set. On the next clock after this, the first flip-flop will toggle to zero (clear), causing $\overline{CE}$ to return high. The next clock clears the second flip-flop, which completes the sequence.

A timing diagram of this circuit is shown in Figure 7. Note that a $\overline{CE}$ high time of approximately two clocks is provided, allowing for false memory-cycle operation.

The $\overline{WR}$ output of the 80186 becomes active too early to allow for a leading-edge write. This situation is remedied by gating $\overline{WR}$ with the $\overline{Q}$ output of the second flip-flop, which delays it long enough to meet the 2186 write requirements. Note that the rising edge of $\overline{WE}$ is timed such that a trailing-edge write is also allowed for full SRAM compatibility.

8088/2187

The circuit in Figure 8 interfaces the synchronous 2187 to an 8088 in such a way as to make the iRAM refreshes invisible. This method, which is commonly used in many microprocessor/DRAM systems, allows the iRAM to be refreshed every time an OP Code fetch is performed. OP Code fetches are restricted to some memory other than the iRAM (typically EPROM or ROM) because the 2187 cannot be accessed during a refresh cycle. To meet the 2187 iRAM refresh requirements, at least 128 OP Code fetches need to be performed in every 2-ms period, a condition easily met. Certain conditions could, however, jeopardize iRAM refresh. These conditions would include extended-hold (DMA) and WAIT (single-step) states, in which OP Code fetches do not occur. If either of these conditions were allowed to persist for an extended period of time, the contents of the iRAM could be lost.

To synchronize the 2187 refresh with the processor OP Code fetches, the signal M1 is decoded from the processor status bits. This signal will go low at the beginning of any OP Code fetch, thus initiating a refresh in the 2187. M1 then returns high before the OP Code fetch is complete, allowing the 2187 to be accessed on the next cycle if necessary.

Figure 9 shows a two-chip microcomputer system employing the 2186 with an 8051 microcontroller, and Figure 10

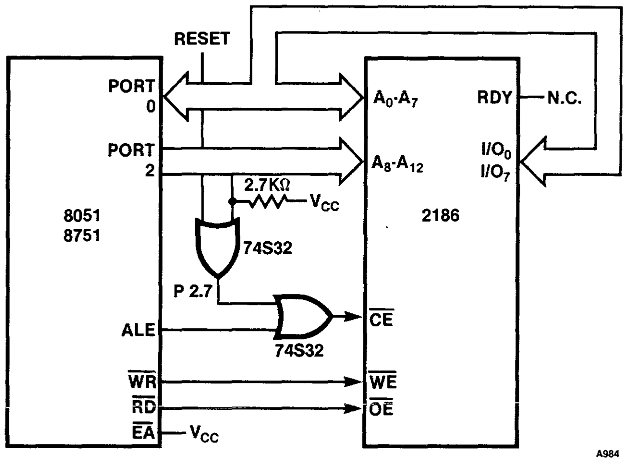


Figure 9—8051/2186 system

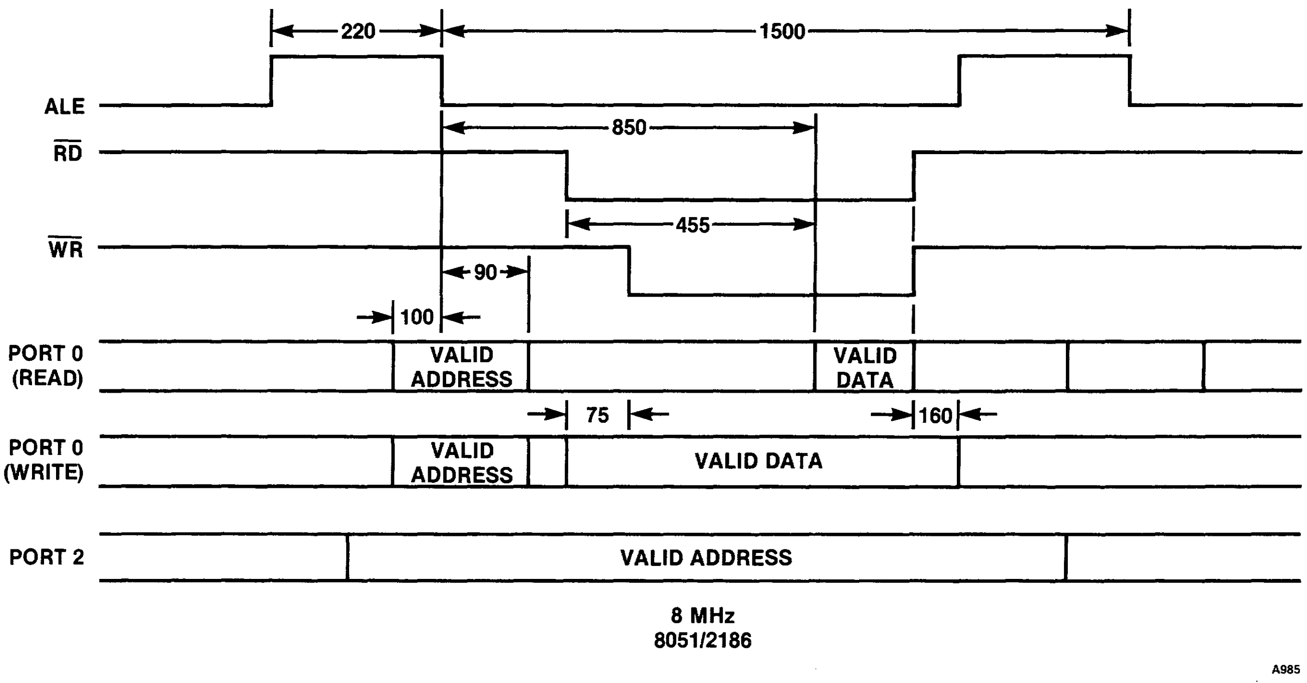


Figure 10—8051/2186 timings

shows timings for this system. The microcontroller includes 4K bytes of program memory on board (EPROM), and the 2186 provides 8K bytes of data memory.

The interface to the multiplexed 8751 bus is very simple. Because the 2186 latches address on the falling edge of $\overline{\text{CE}}$, no address latches are necessary in the system. $\overline{\text{CE}}$ is generated on the falling edge of ALE only for accesses to the lower 32K of external data memory. This is assured by gating ALE with P2.7, which acts as the highest-order address during external memory operations, allowing ALE to generate a $\overline{\text{CE}}$ only when P2.7 is low. To ensure that $\overline{\text{CE}}$s are not generated at other times, P2.7 is initially set to a 1, which it will continue to output until an external memory operation is done. After the external memory operation, P2.7 will return to its preset 1. Because the ports are configured as open drain outputs, a pull-up resistor is included.

The reason for generating $\overline{\text{CE}}$s during external-data-memory operations only is less than obvious. During external-data-memory operations, the 8751 outputs an ALE once every 12 clock cycles, resulting in a $\overline{\text{CE}}$ cycle time compatible with the 2186. The 8751, however, generates external ALEs at all other times also. These ALEs, which occur once every six clock cycles, must be inhibited from generating $\overline{\text{CE}}$s to the iRAM because the 2186 cycle time with wait specification would be violated.

The 8051 does not have a READY input; this does not, however, preclude the 2186 from being used with it. After examining the access-time requirements of 8051 data memory, it can be concluded that for speeds up to 8 MHz, a 2186-25 is fast enough to meet the 8051 memory speed requirements—even in its worst case not-ready condition. This access time requirement would be 850 ns for the 8051 at 8MHz; the 2186-25 worse case access time with refresh is 675 ns.

CONCLUSIONS

The iRAM fulfills all three requirements of microprocessor memory. Using a dynamic RAM memory cell, it meets the density and low-cost criteria. Also incorporating complete onchip control, the iRAM satisfies the ease-of-use and flexibility requirements. Several examples of microprocessor systems have been shown to demonstrate the ease of use. For memory sizes of 8K to 64K bytes, no other memory device simultaneously satisfies all microprocessor memory requirements as does the iRAM.

MULTIBUS® continues to evolve to meet the challenges of the VLSI revolution

by STEVE COOPER
Intel® Corporation
Hillsboro, Oregon

ABSTRACT

MULTIBUS is the world's most popular microprocessor system architecture. The success of the MULTIBUS is attributable to its compatible evolution, always meeting the demands of new VLSI microprocessors without forsaking compatibility with existing products. New system architecture's need for the use of large amounts of high performance memory is the motivation for continued MULTIBUS evolution. This need is addressed through the introduction of the Local Bus Extention (iLBX™). Whereas the local bus under the previous definition was physically limited to on-board execution, the iLBX evolution allows the local bus to span up to five separate boards. The benefit of the LBX is the ability to achieve on-board performance when operating out of physically separate boards. Intel's iSBC® 286/10 single board computer combines the latest VLSI with the iLBX extension, creating a new level of microcomputer system performance. This new board vividly demonstrates that the MULTIBUS continues to evolve to meet the challenges of the VLSI revolution.

BACKGROUND

The MULTIBUS system architecture was developed in 1975 by Intel Corporation to make microprocessors easier to use. The original MULTIBUS contained three unique qualities that have differentiated the MULTIBUS systems from traditional computer systems.

The first of these qualities is its standardization. The MULTIBUS specification (IEEE 796) is precise enough so that MULTIBUS boards from different vendors are fully compatible. The second MULTIBUS quality is multiprocessing—the ability to have multiple CPU boards in one system. This ability is the basis for distributed processing and allows a complex design to be built from easily managed modules. The third quality is the MULTIBUS' generality. The architecture is designed to accommodate any and all microprocessors, and all popular CPUs have been adopted to the MULTIBUS. Intel has encouraged other vendors to use the MULTIBUS by providing specifications, applications notes, and even bus interface ICs.

The popularity of the MULTIBUS has been phenomenal. Over two million MULTIBUS compatible boards are in use today. There are now approximately 120 vendors providing 1,300 different MULTIBUS compatible products. The advantages of such a widespread standard acceptance can be summarized as better products for less cost.

AN EVOLVING STANDARD

The MULTIBUS standard has evolved many times to accommodate the rapidly advancing microprocessors. The long-term success of the MULTIBUS is attributable to its compatible evolution, always meeting the demands of new VLSI microprocessors without foresaking compatibility with existing products. MULTIBUS evolution has been both structural and architectural. Structural evolution has allowed the bus to support new CPU capabilities (Adding address lines to accommodate 24-bit addressing). Architectural evolution (see Figure 1) enhances the MULTIBUS design philosophy of functional partitioning within loosely coupled distributed processing systems.

The VLSI revolution is continuing with even more powerful microprocessors on the horizon. These microprocessors present new challenges to the MULTIBUS. The most pressing challenge is the architectural need for large amounts of high performance memory. As in the past, the MULTIBUS has evolved to meet this need while maintaining compatibility with the existing MULTIBUS family.

LOCAL BUS EXTENSION

Microcomputer system memory size requirements have grown faster than VLSI technology. Applications were rarely larger than 32 kilobytes four years ago; applications of the future will utilize several megabytes of memory. This 100 times increase in memory demand has occurred over the same period as a 16 times increase in memory chip density. The result is that more board space is required for microprocessor system memory than ever before (see Figure 2).

As the need for more memory was mounting, the performance disparity between on-board (or local) memory and

1975	First MULTIBUS® board (iSBC® 80/10 board)
1978	Dual Port Architecture Improves Performance and Reduces Cost
1978	Extensions Added to Support Compatible 8- and 16-Bit Operations
1979	Intelligent Slaves Enhance Function/Partitioning
1980	iSBX™ Bus Added For On-Board Flexibility
1981	Addressing Extended to 24-Bits
1982	MULTICHANNEL™ Bus Added to Standardize DMA Interfacing
1983	iLBX™ Bus Added For Large, High Performance Memory Access

Figure 1—MULTIBUS evolution

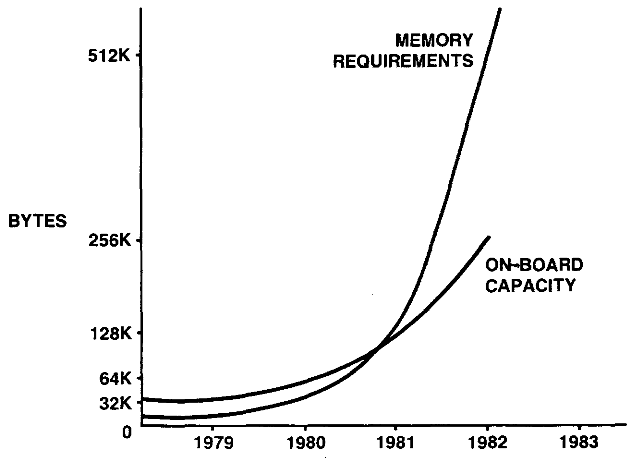


Figure 2—Microcomputer memory requirements have outpaced on-board capacity increases

off-board memory was growing (see Figure 3). In 1975, processor boards ran equally fast from off-board memory as from local memory. Today's high performance processor boards, however, run (worst case) two times faster in local memory than in off-board memory. This performance disparity has lead MULTIBUS board vendors to design boards with increasing amounts of local memory. The demand for even larger amounts of local memory combined with the physical board space limitations is the motivation for architectural evolution of the MULTIBUS.

The local bus extension to the MULTIBUS architecture extends the concept of the local bus. Whereas the local bus under the previous definition was physically limited to on-board, the iLBX evolution allows the local bus to span up to five separate boards. The benefit of the iLBX extension is the ability to achieve local performance when operating out of physically separate boards.

The iLBX extension not only provides tremendous performance benefits, but also maintains compatibility with previous MULTIBUS products. The iLBX physical connections are on the P2 connector of the MULTIBUS card. The lines of the P2 connector used for the iLBX were previously declared *Reserved, not bussed.* But even custom board designers who have used these lines need not worry. The iLBX concept groups a CPU board with several memory boards into a *virtual SBC* (see Figure 4). Therefore only the logical group that supports the iLBX is connected together. The full iLBX specification is publically available today, and the first boards that support the iLBX extensions are now being shipped by Intel Corporation (see Figure 5).

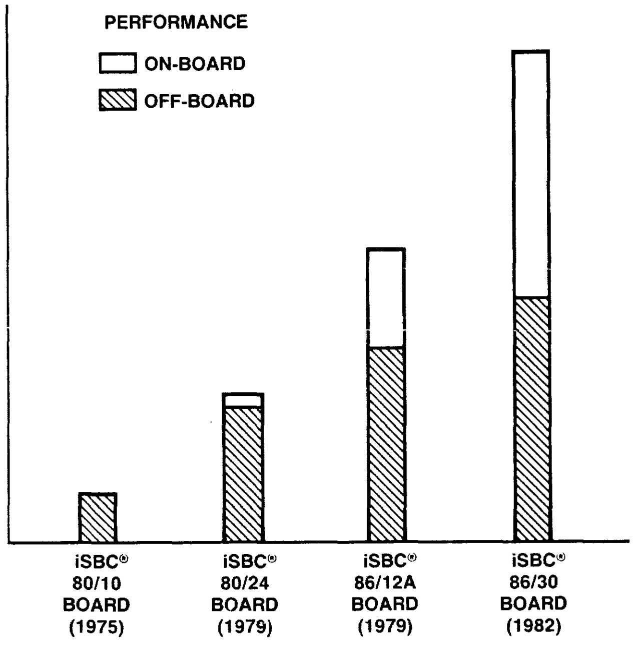


Figure 3—Performance from on-board memory has increased faster than performance from off-board memory

THE LATEST PRODUCT

The iSBC® 286/10 Single Board Computer combines the most advanced VLSI microprocessor with the enhanced MULTIBUS architecture. The performance of the combination is awesome. The iAPX 286 CPU is Intel's latest 16-bit microprocessor implemented with 130,000 transistors on one chip. This CPU performs many internal functions, including memory management and protection, in parallel thus executing instructions in less clock states. By using less states for execution, the iAPX 286 CPU accentuates the effect of memory overhead, known as wait states.

The iLBX bus provides the architectural solution by allowing iLBX memory performance to equal on-board performance. A fastest instruction analysis of the high performance iSBC 86/30 board versus the iSBC 286/10 board vividly illustrates the combined effects of the iAPX 286 and iLBX combination (see Figure 6).

The fastest instruction analysis shows the performance advantage of the iAPX 286 CPU as 2 to 1 and the off-board memory overhead reduction (due to the iLBX architecture) as 3.5 to 1. The combined effect of these two positions the iSBC 286/10 in a new performance class for microcomputers.

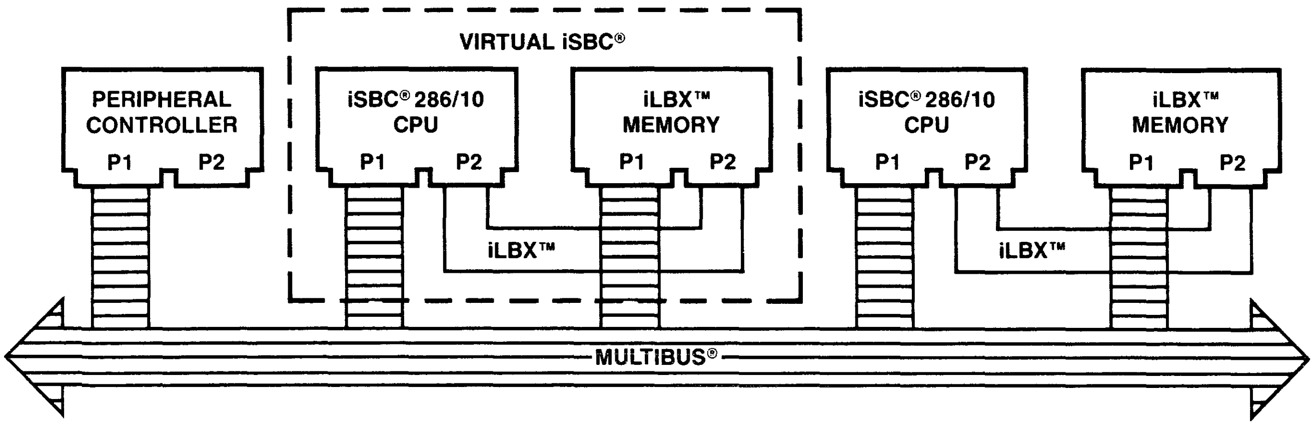


Figure 4—MULTIBUS®/iLBX™ configuration

iSBC® 286/10	— iAPX 286-based CPU board
iSBC® 028CX	— 128K byte ECC RAM board
iSBC® 056CX	— 256K byte ECC RAM board
iSBC® 012CX	— 512K byte ECC RAM board
iSBC® 428	— 16 site JEDEC 28-Pin Site board
iSBC® 580	— MULTICHANNEL™ to iLBX™ DMA controller

Figure 5—iLBX™ supporting products

	# OF CLOCK STATES (@ 8 MHz, ONE STATE = 125 ns)	
CPU INSTRUCTION	iSBC® 86/30 BOARD 4	iSBC® 286/10 BOARD 2
MEMORY OVERHEAD	2 - 7 (ON-BOARD) (OFF-BOARD)	2 - 2 (ON-BOARD) (OFF-BOARD)
TOTAL	6 - 11	4

Figure 6—iSBC 86/30 vs. iSBC 286/10 fastest instruction analysis

CONCLUSIONS

New products like the iSBC 286/10 board prove that a standard architecture can successfully evolve to meet new challenges. The MULTIBUS standard is the world's most popular microprocessor architecture greatly due to its compatible evolution. Future needs such as the accommodation of 32-bit microprocessors will necessitate the next evolution of MULTIBUS. Making changes to an industry standard is a difficult, but in this case, essential task. Each change runs the risk of destroying the MULTIBUS standard, but if managed correctly actually strengthens it. The MULTIBUS has endured many challenges since its inception and must continue to meet the new needs of the VLSI revolution.

Analysis of the M6809 instruction set

by JOEL BONEY
Motorola, Inc.
Austin, Texas

ABSTRACT

The M6809 has now been in the marketplace for about 3 years and is one of the most popular midrange microcomputers. With 3 years of history, it is now possible to analyze many of the existing M6809 programs to see how the computer is actually used.

This paper includes data I took regarding instruction-set and addessing-mode usage on existing M6809 programs.[1] The specific information should be of interest to M6809 programmers and to future computer architects who wish to create similar machines. Beyond the specific M6809 information, however, there are some basic usage trends that are apparent in almost all Von Neuman architectures. Therefore, the information in this paper will be of interest to most users of microprocessors.

The data point out to programmers and system engineers what attributes of a computer's instruction set really affect the memory efficiency and throughput and what attributes don't matter. With this knowledge the programmer/system engineer should be better able to evaluate a microcomputer before he selects one for his project.

INTRODUCTION

In the spring of 1977 several of us at Motorola felt it was time to plan the follow-on part to the successful M6800 microprocessor. We were not the first to envision such a part, but we were the first to actually have the time and resources to proceed with the design. The new part was labeled the M6809. Terry Ritter and I were assigned the task of defining the new architecture.

As part of the preliminary design of the M6809 we did an analysis of the then existing programs written for the M6800. Much of the data we gathered from this analysis was very helpful in the design of the M6809.

Several years have now passed since the introduction of the M6809, and I felt it was time to analyze how the M6809 is actually being used. I hope these data will be as useful to the computer architects that follow us as the M6800 data were to us. Further, I hope these data will enable programmers and system engineers to be more intelligent in their selection of microcomputers for their applications.

GOALS AND CONSTRAINTS OF THE M6809 PROJECT

Every design project in industry begins with some goals that are shared by the designers, the marketers and, hopefully, by the customer. Every project also has some design constraints that must be adhered to in order to design a product that is producible. To understand the analysis of the M6809 that follows, it is necessary to have some understanding of the goals and constraints of the M6809 design project.

A personal goal held by both my coarchitect Terry Ritter and me for the M6809 project was that we wanted to prove that it was possible to produce an inexpensive microprocessor that was also easy to program. We felt that too many of the existing microprocessors were needlessly difficult to program. We suspected that the reason was not that it was impossible to make a microcomputer that was easy to program, but, rather, that the architects of the early microprocessors were generally more hardware oriented than software oriented.

Our experience told us that the consistency (regularity or orthogonality) of the instruction set was one of the features of a computer that made it easy to program. We wanted all the instructions, addressing modes, and system resources, such as registers, to be treated consistently. Analysis of the M6800 showed that instructions such as add *B* to *A* were rarely used despite the fact that they provided a useful function with better than average performance. The reason they were not used was that these instructions were unusual; they behaved differently than other instructions. It was our observation that programmers will not use instructions that are hard to use or that require the programmer to remember peculiarities about their execution.

The second goal of the design team was to support the improvements we saw rapidly taking place in the design of microprocessor software. We wanted the architecture to efficiently support modern block-structured high-level languages. Features such as stack addressing were included for this purpose. We also wanted to better support assembly language with the ability to write recursive, reentrant programs and position-independent programs.

Another goal was to improve significantly the performance of the M6809 as compared to that of the M6800.

Along with goals must come some constraints. We felt that the M6809 must be compatible with the M6800/M6801 at either the assembler-source or machine-code level. The machine-code level was preferable. Because it was impossible, however, to get the necessary throughput improvements and remain machine-code compatible, we selected assembler source-code compatibility.

BRIEF OVERVIEW OF THE M6809 ARCHITECTURE

To understand the data analysis that follows, it will be helpful to have a working knowledge of the M6809 architecture. The following sections describe the programmer's model, the instruction classes, and the addressing modes of the M6809.

Programmer's Model

The M6809 is a $1\frac{1}{2}$-address, 8-bit Von Neuman architecture microcomputer. Figure 1 is the programmer's model of the M6809.

The A and B accumulators are general purpose 8-bit accumulators that can be considered as one 16-bit accumulator for 16-bit operations. When used as one 16-bit accumulator they are called the D accumulator.

The X and Y index registers are general-purpose index registers used in the various forms of indexed addressing. The U and S registers are also index registers, but they have the additional quality that they can be used as stack pointers. The U register is called the user stack pointer. The S register is the hardware stack pointer and is also used by the hardware to store machine state during subroutine calls and interrupts.

The program counter on the M6809 is 16 bits wide, thus supporting an address space of 65, 536 bytes. All addresses on the M6809 are 16 bits wide.

The address field of an instruction with direct addressing on the M6809 is only 8 bits wide. The direct page register is a base

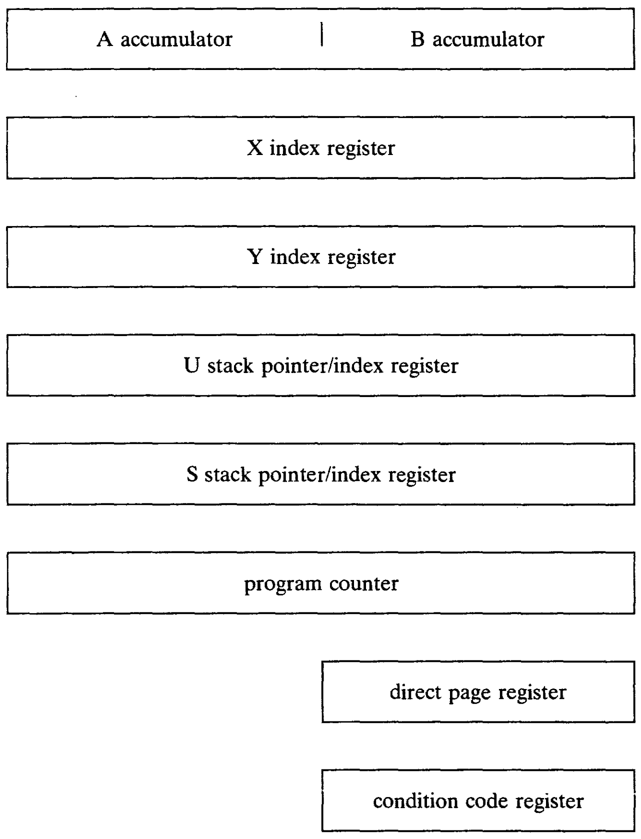


Figure 1—M6809 programmer's model

register that provides the most significant 8 bits of address for direct addressing. The condition code register contains the results from the last arithmetic or logical operation as well as interrupt masks and other control bits.

Instruction Classes

The 6809 has the following seven major classes of instructions:

1. Arithmetic, logical, load and store
2. Read / modify / write
3. Conditional branch
4. Load effective address
5. Push / pull
6. Control transfer
7. Miscellaneous

The arithmetic, logical, and load and store instructions make up the largest set of instructions. They are $1\frac{1}{2}$ address instructions that get one of their operands from memory and the other from an accumulator, and they store the result, if any, in the accumulator.

The read/modify/write instructions read a memory location or accumulator, perform some operation on its contents (e.g., clear, shift, increment), and store the result back to the same memory location or accumulator.

The conditional branch instructions are used for conditional program control transfer.

The load-effective-address instructions evaluate the effective address of an indexed addressing-mode instruction and return the effective address to an index register. This makes the powerful address-calculation hardware already present for indexed addressing available for address manipulation.

The push/pull instructions allow one or several of the registers to be pushed or pulled on the stacks pointed to by the U or S stack pointers. A single push or pull instruction can push or pull from 1 to 8 registers.

The control transfer instructions include the subroutine calls as well as the unconditional jumps and branches. The miscellaneous category includes instructions such as sign extend, no-operation, and transfer register to register. Their addressing mode, if any, is inherent.

Addressing Modes

The M6809 supports a variety of addressing modes. There are seven major types with several subtypes:

1. Inherent
2. Accumulator
3. Register
4. Immediate
5. Absolute
    - Extended
    - Direct
6. Relative
    - Long
    - Short
7. Indexed
    - Constant offset
    - Constant offset from the PC
    - Accumulator offset
    - Auto increment / decrement

Inherent addressing includes those instructions that have no addressing options. Accumulator addressing is similar to inherent except that an accumulator is specified (e.g., *CLRA*, *CLRB*). Some M6809 instructions specify one or several of the registers as the operands (e.g., TFR D,X—transfer D to X). This is called register addressing. In immediate addressing the source operand is assumed to be in the memory location immediately following the current opcode. The M6809 supports both 8-bit and 16-bit immediate values.

In absolute addressing all or part of the absolute memory address is included in the instruction. In extended addressing the full 16-bit address is included in the instruction. In direct addressing only the lower 8 bits of the address are included in the instruction. The upper 8 bits of the address are supplied by the direct page register.

Relative addressing is used for branches. There are both 8-bit and 16-bit relative offsets.

Many of the new features supported by the M6809 lie in its greatly expanded indexed addressing modes. In the constant offset indexed addressing modes a constant value of length 0, 5, 8, or 16 bits is summed with an index register to obtain the effective address used to fetch the operand.

Constant offset from the program counter (program counter relative) works in much the same way except the program counter is used as the index register. This addressing mode is used most often by the load-effective-address instruction to find the starting address of tables in a position-independent program.

In accumulator offset mode the effective address is the sum of the signed accumulator and the specified index register. The original contents of the index register and the accumulator are unchanged by this addressing mode.

In auto increment mode the contents of the specified index register are used as the effective address; then they are incremented by 1 or 2 (postincrement). In auto decrement the contents of the index register specified are first decremented by 1 or 2 and then used as the effective address (predecrement). In both cases the contents of the index register are permanently changed.

All the indexed addressing modes and the extended addressing mode of the M6809 provide for an additional level of indirection. That is, the original effective address calculated by the addressing mode can be used as the address of another 16-bit value that specifies the final effective address.

STATIC VERSUS DYNAMIC ANALYSIS OF ARCHITECTURES

There are essentially two types of analyses that can be performed on an instruction set—static and dynamic. In static analysis, either the source code or the object code of a program or programs is analyzed to determine the frequency of *appearance* of various instructions, addressing modes, registers, and so on. In dynamic analysis, data are taken during the actual execution of a program and are used to determine the frequency of *execution* of an instruction, addressing mode, and so on.

Both types of data are useful for specific purposes. The static data can lead to improvements in future architectures that will reduce the size of the average program. The dynamic data can lead to a reduction in the execution time of the average program. These two improvements are also somewhat related. If a program is smaller, it generally has to fetch fewer bytes of opcode and, hence, runs faster.

STATIC ANALYSIS OF THE M6809

Instruction Classes

To make the data as useful as possible, I analyzed static code from several different classes of programs. I tried to balance the amount of code in each class so that one class of program would not bias the data. I classified the programs into the following classes:

Program Class	*Number of Bytes*
Compiler-generated code	14549
Compiler code	7695
Application code	26305
Monitor code	6293
Numeric code	7135
	61977

Average Instruction Size

One parameter of interest is the average size of an M6809 instruction. The data can be useful when estimating the memory needed for an application. The size of the average instruction for the various program classes and for all classes combined is given in the list that follows.

Class	*Average Size*
Numeric	2.16 bytes
Monitor	2.27 bytes
Compiler	2.30 bytes
Application	2.40 bytes
Compiled	2.43 bytes
All	2.35 bytes

If an M6809 programmer can estimate the approximate number of source lines he will write, he can use 2.35 to estimate his total memory requirements.

Most Frequently Appearing Single Opcodes

There are two ways of looking at the static data. One is to count the percentage of times an instruction (opcode plus additional addressing bytes) appears versus the total number of instructions. I call this the percentage by count. The other is to count the percent of the total bytes actually taken by an instruction. I call this the percentage by bytes.

Table I is the data from the 10 most frequently appearing single opcodes in the concatenation of all of the static data. It is interesting to note that the top 10 opcodes represent 37.4% of all instructions. Since there are 266 possible opcodes in the M6809, these 10 opcodes are only 3.76% of all possible op-

TABLE I—Top 10 most frequently appearing M6809 opcodes

Opcode	Instr.	By Count	%	By Bytes	%
17	lbsr	2307	8.76	6921	11.17
30	leax	922	3.50	2653	4.28
34	pshs	910	3.46	1820	2.94
86	lda immed.	906	3.44	1812	2.92
20	bra	877	3.33	1754	2.83
8e	ldx immed.	862	3.27	2586	4.17
26	bne	804	3.05	1608	2.59
27	beq	800	3.04	1600	2.58
ed	std indexed	739	2.81	1584	2.56
cc	ldd immed.	722	2.74	2166	3.49
	Total		37.40		39.53

codes. Although this may be a surprise to those readers who have never seen instruction-usage data before, it is consistent with most modern architectures.

The top three opcodes are new M6809 instructions that were not available on the M6800. They account for 15.72% of all opcodes. Clearly there was a need for these new instructions.

Most Frequently Appearing Opcodes by Class

Although it is useful to know which individual opcodes occur most frequently, it is more useful to have the data broken down into slightly larger classes. Table II contains the top 10 classes sorted by count for the concatenation of all the static data.

The first 6 classes account for over 52% of all instructions and the top 10 for 66.69%. We can conclude that the M6809 behaves like most computers in that a very few instruction types account for most of the instructions.[2] Furthermore, most of the instructions are in the load-store category. (Pushes and pulls are also classified as loads and stores in some literature.)

Most Frequently Appearing Instructions by Large Class

We can take an even larger view of the instruction classes. These data are useful for comparing the usage of the M6809 to other computers. Table III contains these data along with static data gathered by Leonard Shustek for the IBM 370 and PDP-11.[3]

From the data in Table III we can deduce that the M6809 is not much different from other Von Neuman machines. All three machines have a high percentage of loads and stores, subroutine calls, conditional branches, and compares/tests. Furthermore, the amount of arithmetic and logical instructions is low.

What this tells the system designer who is trying to evaluate the memory efficiency of a microcomputer (remember we are dealing with static data here) is that the overall size of the program will be determined by a few instruction classes. Also, since the loads and stores are heavy users of the addressing modes, the byte efficiency of the addressing modes will greatly effect the efficiency of the architecture.

TABLE II—Top 10 classes of M6809 instructions

Class	Count	%	Bytes	%
16-bit loads	4114	15.62	11291	18.22
8-bit loads	2868	10.89	6144	9.91
Long branch subr.	2307	8.76	6921	11.17
Load eff. addr.	1708	6.49	4629	7.47
Push	1426	5.42	2852	4.60
Store 16-bits	1376	5.23	3155	5.09
Store 8-bits	1219	4.63	2991	4.83
Branch always	877	3.33	1754	2.83
Compare	860	3.27	1792	2.89
Branch not equal	804	3.05	1608	2.59
Total		66.69		69.60

TABLE III—Comparison of static data

Class	M6809	IBM 370	PDP-11
Load/store	50.83	48.00	32.80
Call	12.49	5.50	6.30
Cond branch	10.07	15.30	20.10
Control transfer	5.26	?	†
Cmp/tst	5.63	8.80	6.50
Arith/logical	4.04	3.50*	3.00‡
Other	11.68	18.90	26.30

*Subtract only.
†Control transfer included in conditional branch.
‡Add only.

Static Appearance of Addressing Modes

Another major area of interest is the use of the addressing modes of a computer. Table IV shows the addressing mode statistics for the concatenation of all the static data.

Indexed addressing is by far the most frequently appearing addressing mode because many of the unique features of the M6809 addressing modes are hidden under the umbrella of indexed addressing. For this reason indexed addressing is discussed in more detail in a later paragraph.

The relative addressing modes (short and long) account for 25.01% of all addressing modes. This indicates that M6809 programmers are using the relative rather than the absolute control transfers and subroutine calls. In short, programmers are writing a lot of position-independent codes for the M6809. The relatively small amounts of extended and direct absolute addressing also back up this conclusion.

Indexed addressing static statistics

Since indexed addressing represents about 72% of all the addressing modes that reference memory (direct, extended, and indexed), we now spend some time looking at the indexed addressing data. Table V breaks the indexed addressing down into its subgroups. The basic subgroups are the no offset, the

TABLE IV—M6809 static addressing mode usage

Addressing Mode	Count	%
Indexed	7371	27.99
Immediate	5132	19.49
Short relative	3532	13.41
Inherent	3466	13.16
Long relative	3054	11.60
Extended	1937	7.36
Direct	958	3.64
Accumulator b	456	1.73
Accumulator a	424	1.61
Indirect	175	0.66

Table V—Static indexed addressing data

Addressing Mode	Number	% of total
No offset (offset = 0)	961	13.04
5-bit offset	3940	53.45
8-bit offset	631	8.56
16-bit offset	572	7.76
8-bit offset from PC	92	1.25
16-bit offset from PC	139	1.89
a accumulator offset	85	1.15
b accumulator offset	99	1.34
d accumulator offset	113	1.53
Auto increment by 1	286	3.88
Auto increment by 2	120	1.63
Auto decrement by 1	43	0.58
Auto decrement by 2	273	3.70
Extended indirect	17	0.23
Average additional bytes for indexed =		1.17

constant offset, the register offset, the auto increment/decrement, and extended indirect.

The constant offset varieties account for 72.91% of the total. If no offset is included with the constant offset subgroup, we find that 85.95% of the indexed instructions are of a simple type. The program that took the data also calculated the average number of bytes that are added for each indexed addressing mode above the base opcode. The average is 1.17 bytes. Since the minimum possible is 1.0 bytes, this is a very encouraging statistic. The code-size penalty for providing all the new M6809 indexed addressing modes is minimal. This is good news. As stated previously, the memory efficiency of the loads and stores and the addressing modes has the greatest influence on the memory efficiency of the whole architecture.

DYNAMIC ANALYSIS OF THE M6809

Although the static data used in the previous sections are useful in predicting the *size* of a program, data taken while a program is actually executing are more useful in determining the *throughput* of a computer. Unfortunately, reliable dynamic data are much harder to obtain than are static data; hence, there are few dynamic data for microprocessors.

There are two basic ways of collecting dynamic data. One is to build special high-speed hardware to monitor the instruction execution of a computer. This method has the advantage of being real time but has the disadvantage of being very expensive. To reduce hardware costs, it is usually necessary for the hardware to take only snapshots of a fixed number of cycles. It is hoped that these snapshots will faithfully represent the execution characteristics of the whole program.

The second method is to run programs using a simulator. The simulator can be instrumented to collect the data cycle by cycle. The problem with a simulator is that it slows down the program's execution so much that the statistics may become warped. This is especially true for real-time programs. In fact, many real-time programs won't run on a simulator at all. Furthermore, the simulator may have problems simulating the I/O and interrupt portions of the programs.

I used the simulator method for collecting the data presented in the next sections.

Program Mix

Because of the limitations just mentioned, I was able to analyze only five programs dynamically. The are shown in the list that follows.

Program	*Description*	*Source Language*
Chess	Chess playing program	Assembler
Ed	Line editor	Assembler
Mon	Small monitor	C
Mopet	Automatic test generator	Pascal, Assembler
M6839	Floating point package	Structured Assembler

Although I would like to have analyzed more programs, I feel these programs are representative. However, I did not feel justified in concatenating all the data into one data set as I did with the static data. In the following sections the dynamic data are presented independently for each simulated program.

The number of instructions and cycles in the simulation for each program are the following:

Program	*Instructions*	*Cycles*
Mopet	451,131	2,015,244
Chess	385,698	1,797,514
M6839	163,370	719,976
Ed	149,521	702,876
Mon	86,406	477,887
Total	1,236,126	5,713,497

Cycles Per Instruction and MIPS

A metric of interest for a processor is the number of cycles taken by the average instruction and the millions of instructions per second (MIPS) for each program. These data are contained in the list that follows.

Program	*Avg. Cycles/Instr.*	*MIPS at 2 MHz*
M6839	4.41	.454
Mopet	4.47	.447
Chess	4.66	.429
Ed	4.70	.425
Mon	5.53	.362
Average	4.75	.423

The data are fairly consistent, and a MIPS rate of approximately .4 can be used by a programmer to successfully estimate the execution speed of most M6809 programs.

A note on MIPS is in order here. The actual amount of work (throughput) done by a computer is a function of both the MIPS and the size or amount of data actually operated on during each instruction. For example, a 1-MIP 32-bit machine will have about four times the throughput of a 1-MIP 8-bit machine. The M6809 is somewhere between an 8-bit machine and a 16-bit machine.

Most Frequently Executed Opcodes by Class

Using the same classes of opcodes as described in the static data, we can determine what classes of instructions are most frequently executed. Table VI is the union of the top 10 classes for each program. Note that it takes 24 separate classes to get the union of the top 10 classes. This indicates that the dynamic data are not as consistent as the static data, where the union of the top 10 would only include 14 separate classes.

Most Frequently Executed By Large Class

Table VII contains the union of the three largest classes for each program. In both static and dynamic frequency, loads and stores make up the largest class of instructions by far. Next in frequency in the dynamic data are the conditional branches. Compares and tests also have a high dynamic frequency. Calls have a high frequency, but not as high as in static. Probably the most surprising result is that in programs that have a lot of shifts to begin with (statically), the dynamic frequency of the shifts is even higher.

Table VI—Union of the top ten classes (dynamic) showing % of executed instructions

Class	Chess	Editor	Monitor	Mopet	M6839
Load 8-bit	19.20	11.87	10.10	8.04	8.76
Branch if equal	9.93	6.56	1.44	12.07	.59
Load 16-bit	7.81	8.80	10.41	7.35	2.05
Load eff. addr.	5.06	4.82	3.72	10.95	2.92
Store 8-bit	4.85	5.01	1.49	2.19	4.22
Store 16-bit	4.23	3.58	3.29	4.26	1.88
Branch if not =	4.21	9.32	2.68	6.99	3.80
Bit test	3.49	0	.59	.32	.01
Branch if minus	3.22	.06	.06	.04	.03
Increment	2.89	.46	1.48	.32	.92
Compare 8-bit	2.74	13.18	6.56	8.68	9.35
Jump to subr.	2.64	6.45	3.29	2.60	.05
Push	2.19	3.37	4.20	1.45	1.55
Return from subr.	1.97	3.04	4.90	2.48	2.30
Pull	1.84	2.49	3.90	.77	.95
Compare 16-bit	1.40	5.62	1.39	11.63	1.25
Decrement	1.21	0	.08	.44	4.97
Branch always	.95	6.47	.64	4.64	6.20
Branch less than	.19	.09	0	.01	5.62
Branch higher/same	.11	.27	.30	3.10	1.68
Branch lower	.07	.41	3.56	1.68	.27
Long branch subr.	.04	.73	5.15	1.78	1.15
Rotate left	.02	.08	.06	.09	10.96
Rotate right	0	0	3.77	0	10.63
Other	19.74	7.32	26.94	8.12	16.01

Table VII—Union of the top three largest classes (dynamic)

Class	Chess	Editor	Monitor	Mopet	M6839
Load	27.01	20.67	20.51	15.40	10.81
Cond. branch	21.81	21.12	9.24	24.76	15.86
Store	9.08	8.59	4.78	6.46	6.10
Cmp/test	6.07	18.97	8.32	22.14	12.63
Call	3.18	7.18	10.66	5.54	2.97
Shifts	0.33	0.16	9.50	0.45	23.33

To a user trying to select a microprocessor, these data indicate that the speed of the loads, stores, and branches will have a very large impact on the throughput. The speed of the addressing modes will directly affect the speed of the loads and stores. Furthermore, the speed of the arithmetic (except compare) and logical instructions is almost irrelevant to throughput.

Dynamic Execution of Addressing Modes

This section presents the dynamic addressing-mode data collected by the simulator. Table VIII contains the frequency of execution of the various addressing modes for the five programs.

In all five programs indexed addressing is by far the most executed addressing mode. In fact, its dynamic frequency is about 10% higher than its static frequency for the same programs. Short relative addressing is a strong second with immediate addressing third. If a future architect were looking to improve the performance of the M6809, it would be advantageous to look at speeding up relative and indexed addressing. Indirect addressing is rarely used.

Indexed Addressing Dynamic Statistics

Since the frequency of indexed addressing is so high, it is worthwhile to see how indexed addressing is being used dynamically. Table IX contains the indexed addressing breakdown for the five programs analyzed. The 5-bit and no-offset indexed addressing are by far the most frequently executed. If it were possible to make these faster, it would certainly im-

Table VIII—Dynamic addressing mode usage

Mode	Chess	Editor	Monitor	Mopet	M6839
Indexed	40.79	33.74	29.76	31.05	41.46
Short relative	22.23	27.48	12.09	30.46	23.78
Immediate	14.23	18.47	15.46	12.12	11.49
Inherent	7.21	9.77	23.19	5.77	6.73
Extended	3.90	8.93	3.29	2.90	0.24
Direct	8.33	0.00	0.00	14.29	0.00
Long relative	1.08	0.88	5.18	1.98	1.27
Accumulator a	1.51	0.09	10.33	0.84	9.07
Accumulator b	0.72	0.64	0.70	0.59	5.95
Indirect	0.69	0.00	0.00	0.04	0.15

Table IX—Dynamic indexed addressing statistics

Addr Mode	Chess	Editor	Monitor	Mopet	M6839
No offset	17.03	23.34	7.47	45.95	10.96
5-bit offset	62.00	39.80	37.82	37.35	62.32
8-bit offset	2.30	0.01	0.00	0.22	2.55
16-bit offset	6.77	0.01	0.00	1.21	0.00
8-bit off. on PC	0.00	0.06	11.21	0.02	0.14
16-bit off. on PC	0.00	0.02	0.00	0.23	1.53
Auto incr. by 1	2.70	34.76	21.26	6.78	1.68
Auto incr. by 2	1.57	1.46	0.00	3.06	0.06
Auto decr. by 1	1.15	0.02	0.05	4.26	0.19
Auto decr. by 2	1.60	0.00	0.00	0.00	0.38
a acc. offset	3.34	0.08	11.06	0.37	3.91
b acc. offset	0.35	0.00	0.00	0.26	15.93
d acc. offset	0.42	0.43	11.13	0.27	0.35
Extended indirect	0.77	0.00	0.00	0.00	0.00

prove the M6809's performance. Auto increment by 1 is used fairly often in an executing program. This is expected since almost all auto increments and decrements are in loops.

SUMMARY

As is the case with most Von Neuman architectures, only a few single opcodes make up a large percentage of all the instructions that appear statically. For the M6809, the top 20 opcodes accounted for over 58% of all the instructions. Three new M6809 instructions headed up the list of the most frequently appearing single opcodes. They were long branch to subroutine, load effective address, and push on the S stack. The rest of the top 20 was composed of loads, stores, branches, compares, subroutine calls, and subroutine returns.

In larger classes of instructions, the following statistics were the approximate static values for the top five classes:

1. Loads and stores	= 36%
2. Subroutine calls	= 12%
3. Conditional branches	= 10%
4. Pushes and pulls	= 8%
5. Load effective address	= 6%

The arithmetic and logical instructions occurred infrequently.

Thus, the loads, stores, subroutine calls and conditional branches are a better metric of memory efficiency than are the arithmetic and logical instructions.

The static addressing-mode data indicate that the most common addressing mode is indexed (30%), followed by relative (24%), and by immediate (20%). The number of direct and extended instructions combined was only 10%. Indirect was practically never used.

In the static indexed addressing data we find that 5-bit and no-offset indexed account for 66% of all indexed instructions. Including the 8-bit, 16-bit, 8-bit program counter relative mode and the 16-bit program counter relative mode, we find 86% of the indexed instructions are constant offset or have no offset.

This indicates to future architects that having several efficient, simple offset indexed forms is beneficial. The M6809 has six such forms as compared to the M6800's one.

The average number of bytes added for each indexed instruction is 1.17 bytes.

There was a larger variation in the dynamic data. There are two reasons for this. One is that I had fewer dynamic data points. The other is that dynamic data seem to be more dependent on the application and programmer style.

The average number of cycles for an M6809 instruction is approximately 4.75. This gives a throughput of .423 MIPS with a 2-MHz M6809.

By classes, the conditional branches all combined to form the second most-executed group, second only to the loads and stores. The other large classes were compare and test, the calls, and the shifts.

In dynamic execution the indexed addressing mode accounts for approximately 35% of all addressing modes. Short relative is about 25%, and immediate is 15%. Long relative addressing usage is fairly low. Indirect is the big loser again.

In indexed addressing, the offset varieties accounted for 72%. This is down from the static data, but is still impressive. Auto increment and the accumulator offsets make up most of the rest of the indexed data.

The dynamic data reinforce the conclusions from the static analysis that a good measure of the overall efficiency of an architecture is best found in the loads, stores, conditional branches, subroutine calls, and addressing modes. This coincides with the modern view that computers spend most of their time in moving data around and making decisions based on the data rather than in number crunching.

REFERENCES

1. Boney, Joel. "Analysis of the M6809 Instruction Set," Report, University of Texas Computer Science Department.
2. Stone, Harold S. (ed.). *Introduction to Computer Architecture*. Science Research Associates, 1975, pp. 525–528.
3. Shustek, Leonard J. "Analysis and Performance of Computer Instruction Sets," Stanford Linear Accelerator Center Report No. 205, STAN-CS-78-658, January 1978.

Tales from the trial trail: Videotex progress in the United States

by GARY H. ARLEN
Arlen Communications Inc.
Washington, D.C.

ABSTRACT

With about three dozen experiments and services under way or in preparation, the U.S. videotex industry is in the midst of lively growth. A number of factors will affect the immediate prospects for commercial services, especially the break-up of the Bell system and the activities of the resulting companies. In addition, the development of cable TV and the creation of hybrid interactive services using other new technologies will affect the growth of videotex and teletext in the U.S. Moreover, pending legal issues may shape the development of videotext services.

INTRODUCTION

Videotex and teletext in the U.S. are about halfway through their incubation period. The first round of experiments has been completed; the pioneering commercial ventures, such as CompuServe and the Source, are refining their activities based on experiences during the past three years. As the second round of trials begins and with an eye toward new commercial ventures, the infant videotex business looks to 1985 before it becomes a viable business venture. Of course, a number of other services and projects will have spurted ahead by that time. Indeed, the development of videotex is on an erratic timetable because of the variety of projects being tested and proposed independently by a wide range of organizations. Moreover, since there is no unified, single approach to these services—nor any consensus about how they should be developed—many questions continue to loom over the upbringing of the business.

Even the terms *videotex* and *teletext* continue to be the subject of debate and individual interpretation. Generically, these interactive information retrieval services are often referred to collectively as *videotex*. But that term also refers to the two-way, fully interactive system offered via telephone lines or two-way cable TV systems. *Teletext* is a one-way system, with a far more limited database, transmitted via digital encoding within a broadcast or cable TV signal. Both services offer alpha-numeric messages and some form of computer graphic images, which the user can call up through a keypad. In the most sophisticated videotex systems, online transactions, including shopping, banking, or electronic mail, can be accomplished, often by gateway connections through the videotex system into a remote computer maintained by a third party.

Table I offers a portrait of the fragmented videotex and teletext universe in the U.S. In addition to the services listed here, several other test projects have already been completed. Moreover, the public projects listed do not reflect the handful of private/closed user group videotex trials being explored internally at an increasing number of companies. In Table I, (T) indicates a test, generally with no fee to users; (C) indicates a commercial service.

The development of the above-listed projects—plus the inauguration of services by still other companies—is taking place amidst continuing debate about factors that will affect the growth of the industry. Most significant is the continuing battle over technical standards, which shows little sign of abating.

With its multinational heritage, videotex has generated endless controversy about standards. The issue of technical standards—especially whether a format based on British, French, or Canadian technology should be used—will remain a stumbling block for the immediate future. The development in 1981–82 of a so-called "North American Standard" (based on Telidon and Antiope formats) has not been universally accepted. Indeed, the U.S. delegation to CCITT has suggested the creation of a Unified Presentation Layer (UPL), which will require a new microprocessor chip able to accommodate the European and North American formats. There is no indication when such a device will be available. Meanwhile, the Federal Communications Commission is encouraging a marketplace approach to technical standards. That means no formal technical standard will be adopted for the U.S., allowing individual companies to sell whatever format they prefer. This encourages grueling competitive efforts as different camps seek affiliations with companies that can help create a customer base for their system. In the teletext area, for example, this has led to promises by two major national broadcast networks (CBS and NBC) that they will use the North American Broadcast Teletext Standard for their transmissions, which are beginning in 1983. Meanwhile, a major cable-satellite network (Satellite Syndicated Systems) is developing a national Keyfax service using British teletext technology. This, of course, could lead to substantial problems if customers in a single location seek data from different sources using incompatible technology. Hence, the uncertainty about technical standards is likely to continue for the next few years.

NATIONAL OR REGIONAL; PRIVATE OR PUBLIC

The nature of the North American market suggests that the most valuable electronic services will be ones that cater to specific audiences—even if the only thing which binds them is a common geography. That means regional videotex and teletext services are most likely to be implemented, although some national services will be created. Many major newspaper and broadcasting companies are developing databases for their local and regional markets. This seems to be a sensible approach since most media companies have extensive resources to develop advertising and information from their current service territories. The viability of such regional services is already being tested and in one case there is evidence that the business is not there, at least not yet. A. H. Belo Co., publisher of the *Dallas Morning News,* closed its BISON local electronic publishing experiment in mid-1982, in large part because there weren't enough owners of personal computers in its operating territory to justify the project. Like other companies, Belo suggests that when the number of users equipped with proper terminals increases it will re-enter the business.

TABLE I—Videotex and teletext activity in the U.S. (January 1983)

UNITED STATES	
Current Services	*Number of Users*
The Source	24,854 (C)
CompuServe	34,100 (C)
Dow Jones	54,000 (C)
CBS-AT & T Videotex (New Jersey)	100 (T)
ContelVision (Virginia)	100 (T)
Pronto (Chemical Bank, NY)	300 (C)
HomeBase (Citibank, NY)	100 (T)
Shawmut Bank of Boston	100 (T)
Chase Home Banking (Chase Manhattan Bank, NY)	100 (T)
MacroTel, Inc. (Empire Bank, Buffalo)	12 (T)
BankShare (Huntington Bank, Columbus)	—
Day and Night Video Banking (First Interstate Bank, LA)	200 (T)
FirstHand (First Bank Systems, Minneapolis)	285 (T)
Bank-at-Home (Financial Interstate, Knoxville)	600 (C)
INDAX (Cox Cable)	
San Diego	500 (T)
Omaha	100 (C)
Times Mirror Videotex (Palos Verdes & Mission Viejo, CA)	350 (T)
StarText (Ft. Worth)	171 (C)
Electronic Editions (Spokane, WA)	140 (T)
Harris Electronic News (Hutchinson, KS)	58 (C)
AgVision (ELANCO, Indianapolis)	850 (C)
Louisville Courier Journal (Louisville)	45 (T)
A-T Videotext (Tiffin Advertiser Tribune, Tiffin, OH)	30 (C)
Time Video Information Services (San Diego and Orlando)	400 (T)
Keyfax National Teletext	350 (C)
WETA-TV (Washington)	10 (T)
WKRC-TV (Cincinnati)	40 (T)
KPIX (San Francisco)	75 (T)
KIRO-TV (Seattle)	21 (T)
WGBH-TV (Boston)	25 (T)
KSL-TV (Salt Lake City)	5 (T)
San Francisco State University	44,000 (C)
TeleCaption (NCI/Sears)	60,000 (C)
Future Services	*Scheduled Operational Date*
Viewtron (South Florida)	Summer 1983
Viewcom/Grassroots (Bakersfield, CA)	1983

The development of regional services leads to the issue of other kinds of special-interest videotex systems, specifically business-oriented ones for closed user groups. The issue of private or public videotex has not yet surfaced in the U.S. to the degree it has in Europe, although a number of companies (such as Videodial [using French format], Rediffusion [promoting British technology] and ModComp) have recently entered the U.S. market offering turnkey private videotex systems. IBM's entry into the U.S. videotex market came in the form of a private viewdata system, based on its Series I minicomputer. It is difficult to foresee at this stage how effective such private systems will be, especially when the concept of videotex is still being defined to corporate customers.

BEYOND BELL

American Telephone and Telegraph Company's divestment of its local operating companies and its new-found freedom to enter competitive business activities—notably data processing—will inevitably change the nature of the U.S. telecommunications industry. Because AT & T has plans involving several aspects of videotex, the company's future activities will play a great role in shaping the development of the business.

The spin-off process will last until 1991, a vital period in the development of videotex. AT & T has already begun to unveil its videotex products, including the highly regarded Frame Creation Terminal. Bell and its soon-to-be-divested local affiliates are working on distribution systems that offer voice-under-data transmission capabilities, ideal for simultaneous use of telephone service for conversation and videotex.

Typically, AT & T is not proclaiming its precise plans as the divestiture process begins. Through its participation in several videotex efforts to date (notably, the Viewtron service with Knight-Ridder Newspapers and the Venture One project with CBS), AT & T is gaining expertise in all aspects of operating a videotex service. It is possible that when the legal restrictions are lifted in a few years, AT & T will expand its role into information services. Meanwhile, the restrictions to keep AT & T out of electronic information may prompt newspaper publishers, who once feared Bell's presence and lobbied hard against it, to drop their own electronic activities. Now that Bell is barred from such services for the time being, the publishers may not feel motivated to pursue any efforts of their own in this area.

THE ROLE OF RIVAL TECHNOLOGIES

Even as videotex and teletext struggle to find a niche in the U.S. marketplace, other technologies, as well as hybrid services involving videotex, are surfacing. To many experts in the data processing and communications business, all of these services are the raw materials with which to build entirely new ventures. For example, the traditional concept of broadcast teletext (vertical interval transmission of news and information) is being amended by forays into full-bandwidth data transmission using video channels on cable TV systems or satellites; Time, Inc., is particularly active in this area.

Other data transmission efforts involve piggy-back projects in which text/data services are carried on the subcarrier frequencies of radio stations or satellite signals. An example of this is a service being proposed by National Information Utilities Corp. and National Public Radio. The primary objective is to offer local one-way data communications to business offices seeking an alternative to expensive "last mile" delivery—a service that will become more expensive as local telephone rates increase. Local data will be carried via a subcarrier on FM radio stations; for clients who need national distribution of data, the nationwide public radio satellite network will provide interconnection.

Several other types of transmission systems can also deliver similar one-way service. For example, low power TV—a broadcast service with authorized transmissions within only, approximately, a 15-to-20-mile radius—is often mentioned as a medium for such local data distribution, possibly using teletext-like facilities. In addition, Multipoint Distribution Service, a local microwave transmission facility, is being promoted for its wireless cable potential. (It should be noted that the development of both these technologies is still in the hands of the Federal Communications Commission.)

Meanwhile the cable TV industry, which now reaches about 34% of U.S. homes, is increasingly attracted to data and text services. Almost all major city cable systems now being built offer substantial capacity for interactive services. During the late 1980s and 1990s, many of the cable systems built 20 years ago will be upgraded to include advanced interactive capabilities. At the same time, the cable industry is beginning to interconnect systems serving contiguous cable operations.

SHOPPING AND BANKING

A great deal has been written about the prospects for electronic at-home shopping and banking on the new systems. Indeed, more than a dozen U.S. financial institutions are already taking part in advanced tests and many more companies are eyeing such participation. Equally important, the shifting regulations and corporate objectives of U.S. financial companies augur an even greater move into such services during the mid-to-late 1980s. In particular, a number of national banking regulations that have so far restricted home banking efforts will be changed. At the same time the financial realities of the early 1980s have resulted in the creation of new financial empires—businesses that transcend the traditional concept of locally owned banks. A number of nationwide financial service companies are being created and their expertise and capital resources will encourage the growth of the teleservices industry. Moreover, the role of companies such as Sears will increase. Sears, for example, seeks to coordinate the real estate and brokerage services it recently acquired, blending them into its existing insurance, banking, and retailing activities. Add to this Sears' ability to sell and service consumer electronics products and it becomes a mighty force for teleservices.

Elsewhere on the merchandising scene, retailers and direct marketing firms are escalating their efforts into electronic selling. Many of them are still awaiting technical refinements, such as the ability to offer video-on-demand to prospective customers. There is, however, lively interest in the possibilities of electronic shopping, although most merchandisers are still demanding proof that the service will result in cost efficiencies before they commit themselves to any further development.

CONCLUSION

The videotex experiments and services in the U.S. have had mixed results to date. The trials generally indicate that information alone is not enough to sustain services, that viewers require additional features, especially transactional services. The precise nature of those services, and the mixture, are being refined by other projects.

Meanwhile, the onrush of alternative services and delivery systems will continue to affect the development of videotex and teletext. The de facto decision to determine technical standards in marketplace competition rather than by government fiat will mean that it takes longer to settle on a format. In turn, that leads to further delays in introducing services to public and private users and in convincing them to invest in such services.

Despite these factors, the parents and guardians of the U.S. videotex industry continue to have great hopes for the healthy development of their child. During the current incubation period, a variety of modifications are inevitable. By the mid-1980s, when the business reaches its next stage of development, some of those questions will have been answered.

Videotex and teletext in the business/consumer marketplace

by LARRY T. PFISTER
Chairman, Videotex Industry Association
New York, New York

ABSTRACT

The impact of interactive television in the marketplace will change our perception of the home television set from that of an entertainment medium to that of an informational mechanism at both the consumer and business levels. A review of the development of videotex and teletext in the United States and abroad highlights significant contributions of the various technologies shaping this rapidly evolving industry. As reflected by the rapid growth of the U.S. Videotex Industry Association, the potential of videotex is being tracked by a broad spectrum of interested parties—from the individual PC user to the corporate board at AT&T. The videotex revolution will profoundly change our lifestyles and will bring instant information-on-demand into our homes and offices.

I appreciate the invitation to participate in the 1983 National Computer Conference and particularly this opportunity to discuss one of my favorite subjects—the future of television. I believe *interactive television*—videotex and teletext—to be the future for television in the United States and around the world. Why should I talk about television at a data processing conference? Primarily to drive home the point that the separate worlds of data processing and video are rapidly converging. Major U.S. data processing vendors are obviously serious about videotex and are gearing up for the U.S. market. Suppliers, including IBM, DEC, Perkin-Elmer, Computer Automation, ADP, Amdahl, Atex, Hewlett-Packard, Tymshare, and nearly 150 other firms, are marketing videotex systems and services in the U.S.

I have been asked to give an overview of videotex and teletext and to discuss briefly some of the activities here and abroad in both the business and consumer marketplace that may affect you as a data processing professional and as an individual citizen. This is a pretty ambitious aim for a short session. Videotex is difficult to define in 25 words or less, and it has been said that teletext is easier to use than it is to explain. Perhaps the best I can hope to accomplish is to hit the high points and suggest that for an in-depth understanding of this rapidly evolving technology, you could contact the Videotex Industry Association or one of our member firms who are pioneering applications in the United States.

If the generic term *videotex* is not familiar to some of you, the names of particular systems, such as Ceefax, Oracle, Prestel, Antiope, or Telidon, are more likely to have a familiar ring. This is no accident, for there are powerful commercial forces (some backed by national governments) hard at work marketing their particular technology to almost anyone who will listen.

The reasons for this are not difficult to discover. It is widely believed that videotex and teletext will become as common as entertainment television in the American household and that the necessary electronics will be directly incorporated inside television sets of the future, permitting the user to send electronic mail to friends, to order goods and services, to obtain information on an almost limitless variety of subjects—both topical (such as news, weather, airline schedules, or stock market information) and nontopical (such as one might find in encyclopedias or do-it-yourself books)—and to play games, either with a computer or with other humans.

There are still a lot of important unanswered questions, such as how quickly the market will grow and, when it does, what the predominant uses of such systems will be. Will telephone companies or cable television systems be the predominant carrier? Is there a danger of system-operator or information-provider monopolies developing? It sometimes seems that these marketplace questions are ignored in the rush of the principal contenders to establish their technology as the standard for videotex or teletext communications. The current situation is reminiscent of the negotiations surrounding the adoption of the NTSC color television standard in the United States and of the subsequent competition between NTSC and the PAL and SECAM standards worldwide.

Clearly, the standards issue is an important one. It would hardly make sense if consumers had to buy several different electronic devices in order to have access to the people and information banks of their choice, or for information providers to have to format their information in a variety of different ways and store them on many different systems in order to approach universal coverage. There are still countries in which there are two competing telephone systems, but such countries are not held up as models of sensible behavior, even by the most ardent free-enterprisers.

In Europe, the issue of standardization of videotex and teletext has been handled with relative ease. In France, both the broadcasting system and the telephone system are government owned and controlled and the same is true in Britain, with the exception that on the broadcasting side there is an independent television authority (although even there it is the content of the television programs that is independent, not the technical standards of transmission). Thus, videotex and teletext standards have been effectively set by government decree or consent in Europe. Perhaps most significant, however, is the fact that these countries have adopted systems in which the videotex and the teletext formats are mutually compatible, thereby making it possible to design inexpensive receivers capable of handling both broadcast teletext and two-way videotex services. And it's far more likely that TV sets will have an inexpensive decoder built inside the set than that they will ever have an outside appendage such as a videodisc or personal computer.

Let me share a few relevant statistics about American television:

1. Today, there are over 80 million television households in the United States.
2. The average American watches television four hours a day.
3. The average U.S. household watches more than seven hours of television a day.
4. The average American spends only 25 to 30 minutes per day on print—newspapers and magazines.

Clearly, with these numbers—if people are in front of the television set for that many hours—there's an opportunity to use the television set for more than just entertainment. The perception of what the television set represents in our homes is changing. In the 1970s, people still thought of TV as a way of receiving just three or four television channels. In recent years, cable television has introduced dozens of channels and all kinds of nonentertainment services. In the United States, Kodak stopped making its 8 mm and Super-8 film products because they, like a lot of others, think that in the future home movies are going to be videotaped and displayed on the TV set. More important perhaps is that electronic games and personal computers, which use the TV set as a display device, are already changing our perception of what the set is.

All of these—cable, pay cable, VCRs, videodisc, personal computers, and video games—are changing our perception of television, which will make it a lot easier to use the TV set as an informational mechanism. On a broad international scale, interactive videotex and teletext systems are being studied, tested in field trials, and introduced as regular services at both the business and the consumer levels. Among the technologies employed internationally are the following primary systems:

1. Prestel (Viewdata) in the U.K. and several other European countries
2. Bildschirmtext in the Federal Republic of Germany
3. Antiope/Didon in France
4. Captain in Japan
5. Telidon in Canada
6. AT & T's PLP in the United States.

Although there are some significant differences between these systems, national and international efforts are under way to develop compatibility for communications and applications interchange. It now appears that the major European systems will adopt a hybrid compatible system known as CEPT and that North America will rally around a proposed standard based on PLP. Furthermore, the U.S. delegation to CCITT recommended the creation of a Unified Presentation Layer, based on open system architecture, to accommodate both the European and North American formats and to set the stage for one global international videotex system that can be used as the telephone system is today.

The fact that compatibility issues are being addressed in a responsible way before videotex and teletext penetrate the U.S. marketplace is a healthy sign. It has taken years for the computer manufacturers to fully appreciate the synergism of compatibility in building a market where the whole is truly greater than the sum of its parts. As a marketer, I would prefer to have a fair competitive share of a large market than all of a tiny one. In your profession, it is probably easiest to think of videotex as perhaps the ultimate extension of the concept of distributed data processing. In the early 1970s, the computer industry realized that using minicomputers to provide distributed processing and information storage (as a complement to a centralized data processing facility) could improve the efficiency and productivity of computerization. Distributed data processing has gained wide acceptance for many applications and it is now the preferred solution in most cases. Until now, users of distributed data processing have been driven by the desire to have an internal information and transaction processing network within a company, particularly when that company is geographically dispersed. The cliche "user-friendly" has come about because the users of distributed data processing often have no knowledge of computer systems.

Digital Equipment Corporation was one of the first companies to recognize the potential of distributed data processing and today commands a major share of that market. Today it's worth noting that DEC already has established itself as a pioneer in videotex. According to the Link Research Division of International Data Corporation, DEC computers are serving as hosts for a majority of videotex and teletext systems worldwide. It's also notable that Digital was a founding member of the Videotex Industry Association in this country.

Videotex becomes the obvious extension of distributed data processing as lay users extend their information requirements to external databanks (such as a stockbroker or bank account) and the computer terminal gives way to the friendly 19" color television set, suitably equipped with an intelligent decoder. The benefits that videotex bring to business data processing are friendly interface and simple graphics systems that can be appreciated by a very wide nontechnical audience and the immediate potential of extending applications into the consumer marketplace. Remember, not every consumer is a businessperson, but every businessperson is also a consumer.

Last fall the Dow-Jones publication *Barron's* ran a cover story entitled "The Videotex Revolution," and called this new medium the last great electronic adventure of the 20th century. The message of that article was clear: The videotex and teletext business is following the same development pattern as telephone, broadcasting, and data processing. Like those technologies, videotex will take about 20 years to mature. Since the British introduced Prestel about 10 years ago, videotex should gain commercial success in the 80s.

Videotex will change the way people make money and the way they spend it. Half of the U.S. work force is already engaged in service-related industries; minicomputers are already enabling many people to tele-commute instead of commuting to the office. Stockbrokers, architects, designers, writers, data processors, and secretaries may perform their jobs using videotex. Retail spending habits will shift once teleshopping in electronic malls surpasses brick and mortar stores in terms of product variety, economy, and convenience. Transportation patterns will also change. There will be fewer trips to banks, stores, and the post office, but more reliance on distribution systems that can process electronically ordered merchandise.

The businesses that accurately predict how these life styles will change can profit from the videotex revolution. The data processing managers who expect to sit around and wait for this new medium to develop and who count on finding pat formulas available at exactly the right moment will find themselves unprepared and outdated.

Winchesters for multiuser/multitask applications

by LARRY JACOB
PRIAM Corporation
San Jose, California

ABSTRACT

The paper addresses the growing need for Winchester disk storage to handle multiuser/multitask applications. The paper reviews key elements necessary for achieving reliable performance, capacity, and fast access (e.g., linear voice coil actuation, closed-loop servo techniques, and processing/packaging methods). High-performance Winchester options for multitasking, how to evaluate them for specific applications, and what the future holds for Winchester technology are also discussed.

This paper was adapted to appear as an article in *Computer Design* magazine, January 1983, an article which is copyright © by Computer Design Publishing Company, 1983.

Winchester disk drives have been in use with large computers for years, but it is only recently that they have been adopted by manufacturers of small business systems, of word processors, and even of personal computers. Unlike large-system Winchesters, however, there is a bewildering variety of small-system Winchester storage products: three different disk sizes; two fundamentally different actuator systems; and a variety of interfaces, ranging from "dumb" to "smart."

In a multiuser/multitask environment, where reliability and performance are essential, Winchester disk drives that provide high capacities and low cost per megabyte are a logical choice. Once the decision to use a Winchester is made, decisions also need to be made regarding the specific type of Winchester and the interface type. These issues are discussed in the rest of this paper.

WHAT ARE THE WINCHESTER CHOICES?

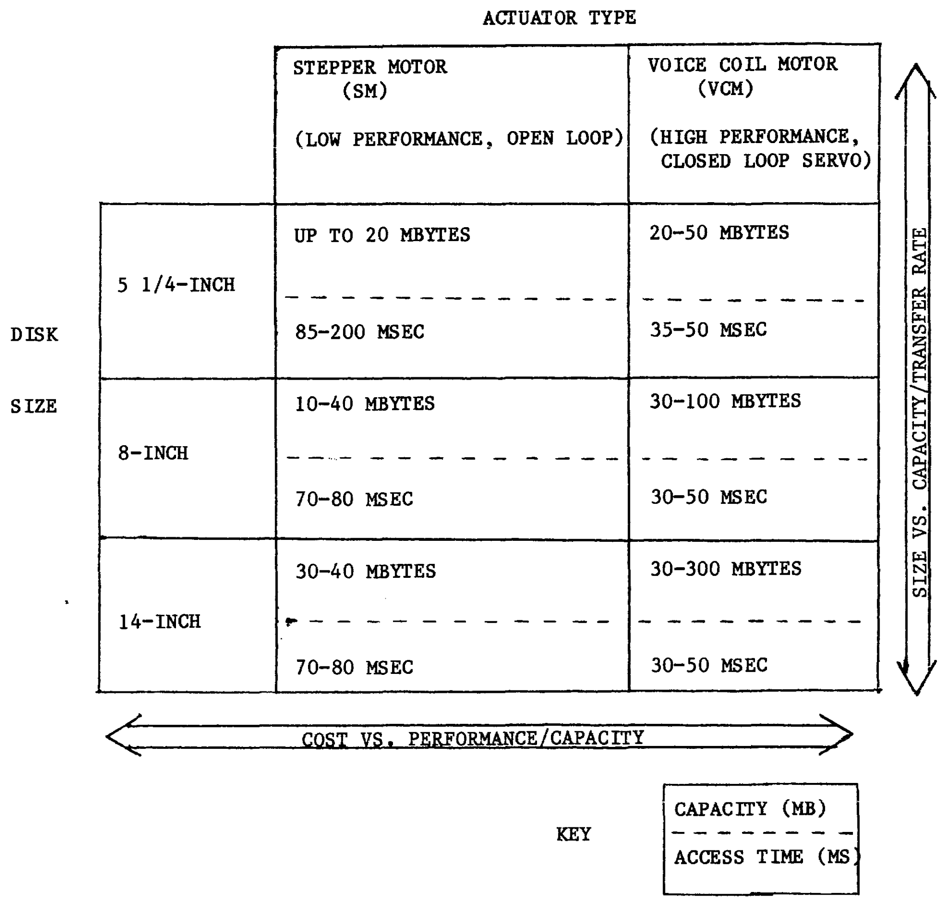
Selecting the right Winchester starts by defining the system requirements. There are three fundamental system types: Single-User/Single-Task (SU/ST), Multi-User/Multi-Task (MU/MT), and Single-User/Multi-Task (SU/MT). Each of these fundamental system types has different basic mass storage requirements, as shown in Figure 1. Networks, or distributed processing, are typically hybrids of the two fundamental system types, and high-powered workstations—frequently part of a network—can be thought of as single-user/multitask (SU/MT) systems. In this system a single user may have several different programs (tasks) working in parallel (the concept of concurrency). These tasks might include performing word processing, compiling a BASIC program, plotting financial information, and sending or receiving an electronic mail message.

Six basic types of OEM Winchester disk drives serve these three system types. They are simply the combinations possible from the three main disk diameters currently available (14-inch, 8-inch, and 5¼-inch) and the two basic actuator systems (the low-performance, open-loop stepper motor and the high-performance, closed-loop voice coil motor). Typical capacity and access time range characteristics are shown in Figure 2. The complex procedure now begins to determine which one (or combination) of the six basic types of Winchester disk drives should be chosen for a particular type of system. The tradeoffs that the system designer makes during this selection process are also depicted in Figure 2.

A mapping of the fundamental system types into the basic drive types (see Figure 3) has been developed to assist in choosing an appropriate type of Winchester disk.

SINGLE-USER/SINGLE-TASK	MULTI-USER/MULTI-TASK	SU/MT
* DEDICATED DATA BASE	* SHARED DATABASE	* DEDICATED/SHARED DATABASE
* UP TO 20 MEGABYTES (MB)	* 30 TO 400 MB	* 20 TO 50 MB
* LOW COST PER UNIT IS KEY	* LOW COST PER MB IS KEY	* BOTH COST PER UNIT & COST PER MB CONSIDERATIONS
* "SLOW" DATA ACCESS ACCEPTABLE	* "FAST" DATA ACCESS REQUIRED	* "MODERATELY FAST" DATA ACCESS REQUIRED
* LIGHT DUTY CYCLE IMPOSED ON ACTUATOR	* HEAVY DUTY CYCLE	* MODERATE DUTY CYCLE
* SMALL SIZE IS KEY: USED IN PORTABLE DESKTOP SYSTEMS, FLOPPY-DISC CAVITIES	* SMALL SIZE IS HELPFUL: USED IN PEDESTAL OR RACK MOUNT SYSTEMS	* SMALL SIZE IS KEY

Figure 1—Mass storage requirements for three popular system types



ACTUATOR TYPE

		STEPPER MOTOR (SM) (LOW PERFORMANCE, OPEN LOOP)	VOICE COIL MOTOR (VCM) (HIGH PERFORMANCE, CLOSED LOOP SERVO)
DISK SIZE	5 1/4-INCH	UP TO 20 MBYTES / 85-200 MSEC	20-50 MBYTES / 35-50 MSEC
	8-INCH	10-40 MBYTES / 70-80 MSEC	30-100 MBYTES / 30-50 MSEC
	14-INCH	30-40 MBYTES / 70-80 MSEC	30-300 MBYTES / 30-50 MSEC

SIZE VS. CAPACITY/TRANSFER RATE
COST VS. PERFORMANCE/CAPACITY
KEY
CAPACITY (MB)
ACCESS TIME (MS)

Figure 2—Six basic types of Winchester disk drives: Tradeoff matrix

THE MAPPING OF THREE SYSTEM TYPES TO SIX DRIVE TYPES

The SU/ST storage requirements of small size and capacity, low cost, and light duty cycle fit the 5¼-inch stepper motor (SM) Winchester best. However, if the SU/ST system is planned to grow into an SU/MT system, the user needs to plan now for using a 5¼-inch voice coil motor (VCM) later. In early 1983, three years after low-capacity (under 10 MB) 5¼-inch drives were introduced, high-capacity 5¼-inch drives in the 40- to 55-MB range have now made their entrance.

The SU/MT storage requirements of moderate capacity and performance with very small size fit the 5¼-inch voice coil motor Winchester best. However, production quantities of these drives are not available, and the product and vendors still need to be field-proved (remember the "8-inch fever" in 1979 and 1980). So those who need capacity and performance in a small package size in 1983 (maybe even 1984) should stick with the 8-inch voice coil motor.

The MU/MT storage requirements of high capacity, low cost per megabyte, and heavy duty cycle fit the 14-inch voice coil motor Winchester best. The main exception occurs when size is more important than large storage capacity. In that case the user should consider the 8-inch voice coil motor, keeping in mind that most 14-inch drives include space for a power supply but 8-inch drives do not. The drive power supply must be considered in space planning if an 8-inch drive is chosen.

Growing capacity needs of the SU/ST and SU/MT applications are reaching into and beyond the range of capacity available in 5¼-inch drives today. However, if the capacity of 5¼-inch drives reaches what is currently projected, and if these drives become available in volume, 5¼-inch drives will eventually replace some of the 8-inch drives in newly designed systems. This will happen first and most furiously at the low-end (open-loop stepper motor) segment of the 8-inch market. Conversely, the portion of large existing MU/MT applications that are becoming size-sensitive will probably grow, leading to the increased use of compact 14-inch voice coil motor, and floppy size 8-inch VCM Winchesters.

Future system storage requirements no longer map directly onto the 8-inch and 14-inch stepper motor Winchesters. This is because the physical size of the Winchesters suits the MU/MT system best, but their access time, duty cycle, and storage capabilities fit them to SU/ST systems best. Thus they map to neither system type, and their use by systems designers will be declining rapidly.

Particular attention needs to be paid to the capacity growth path implications of the device chosen. On the one hand,

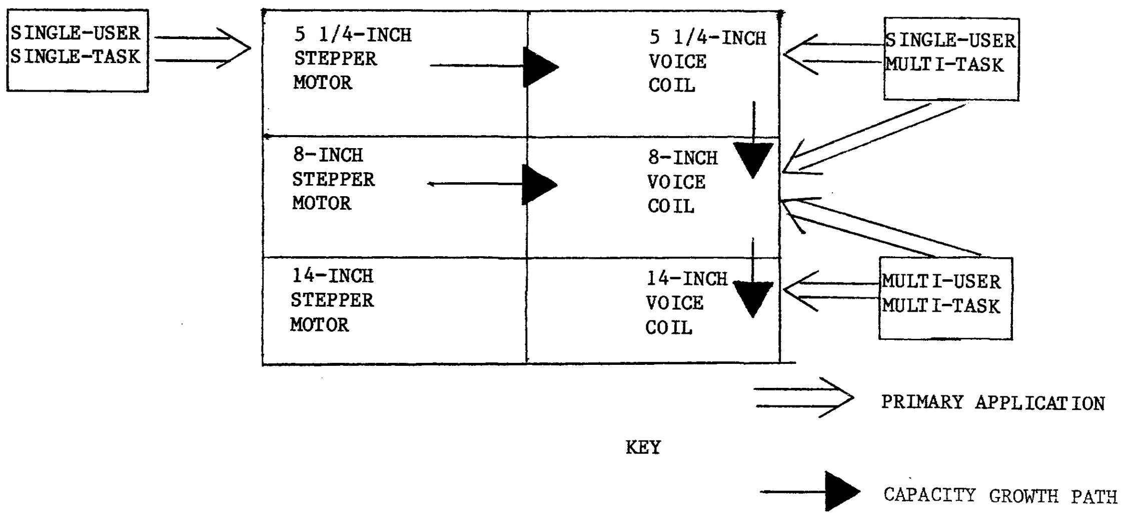


Figure 3—Mapping of three system types to six drive types

there are lateral growth paths, where the physical size of the device remains the same. These are relatively easy to implement without changing interface or systems packaging. On the other hand, vertical growth paths in the same system enclosure, where the physical size of the drive changes, will occur less frequently, because of the capacity increases promised in the 5¼- and 8-inch disk sizes. Nevertheless, these growth paths are important, because of the increasing propensity of systems manufacturers to repackage successful systems for appeal to different markets: Standalones are souped up to serve multiuser markets, and multiuser systems are downsized to serve desktop markets. In these cases, travel along a growth path where the drive interface is common is much easier than if interface, power, and software changes are required.

OTHER CONSIDERATIONS FOR THE MULTIUSER/MULTITASK APPLICATION

Several other issues need to be addressed in the evaluation of high-performance Winchesters: system architecture, system integration, reliability, vendor viability, and total cost of ownership, to name a few.

System Architecture

System architecture can affect drive performance. In MU/MT applications, performance is the key ingredient. It is possible for total system performance to be drastically affected by apparently small changes in disk drive specifications or system timing requirements. For example, a change of a few microseconds in the time required to switch heads could be very significant if it precludes reading the next sector, causing tens of milliseconds to be added to a multisector operation. Similarly, an increase in data rate could cause sectors to be missed because of a suboptimal interleaving factor, again adding tens of milliseconds to each operation.

In evaluating system performance with different Winchester disk drives it is important to verify that the disk is not artificially constrained by a system parameter (like interleave factors) that you, as the system's designer, would be willing to change.

In most disk drive applications a significant part of the total transfer time is spent seeking and waiting. In a well-designed system these times are overlapped and minimized as much as possible. Many variations have minimal effect. When a boundary is approached, however, performance often changes suddenly and drastically.

System Integration

Choosing the Winchester is just the first step. Making it work in a prototype and then in production systems is next. The key to this task is the selection of an interface.

If the system design is brand-new, more decisions are required than if a current model is being enhanced or redefined. In either case, evaluation of the interface options on the basis of functional integrity, specific application requirements (perhaps emulation), controller availability, and the projected longevity of the interface is required.

In today's rapidly changing storage market, choosing the disk drive interface may be a longer-standing commitment than choosing a particular drive capacity level. Projecting the disk requirements for the future of the product line is an important factor. Will the interface choice still be appropriate? In the long term, a high-level interface generally makes the most sense; floppylike, external data separators make the least. In addition, because the market is changing so rapidly, the user should review the possibility of purchasing an intelligent controller, or even a memory subsystem, to accelerate the product to production.

The higher-level interfaces have technical advantages in that they all contain data separators. In truly high-performance products the data separator is a key component in the determination of data integrity. As such, it is to the

OEM's advantage to have the disk vendor responsible for an appropriate data separator design. Another advantage of the higher-level interface is its ability to deliver diagnostic status information.

Interfaces start from very simple and unsophisticated levels that approximate the control capabilities of floppy disk drives. They improve in stages in capability and intelligence until the drive with its interface actually constitutes a disk subsystem in which many sophisticated disk controller functions are added to the data storage capability of the drive. This range is briefly summarized below.

Low Cost/Low Function Interface: Provides the user MFM data- and bit-oriented command and status information. This interface is the most difficult to use, but the most economical from a drive point of view. Therefore it may be attractive to the SU/ST, large-volume, cost-sensitive user with substantial engineering and manufacturing resources capable of designing, testing, and controlling the difficult data separation circuitry required to process MFM data and higher and higher data rates.

Early OEM Disk Interface: This interface provides NRZ data; that is, data separation is performed on board the drive. Commands are byte-oriented, but status is bit-oriented. Control Data Corporation's SMD interface, developed in the early 1970s, has become a standard interface for larger systems. Though most designers consider it technically obsolete, since it is cumbersome to use and costly to implement, the wide range of disk drives available with the SMD interface contribute to its continuing popularity with systems designers.

Modern Disk Bus: This interface has an NRZ data stream like its predecessor; but both command and status are byte-oriented, and the interface is typically microprocessor-driven. The modern disk bus interface costs a little more than the low-cost/low-function interface; but is substantially easier for the system designer to use, and it provides better data integrity, since the data separator circuitry is on board the drive and controlled by the disk drive supplier. Examples of this interface include the ANSI standard and PRIAM interfaces.

Intelligent Bus: Intelligent bus interfaces such as ISI, SCSI, and SMART are indicative of the trend toward dispersing intelligence to components of computer systems and making disk drives easier for system designers to use.

Using a microprocessor to provide broad control capability at low cost, the intelligent bus type of interface is completely byte-oriented and is designed to adapt readily to commonly used microprocessor I/O busses. High-level versions of the intelligent bus include disk formatting and defect mapping, implied seeks, daisy-chaining capabilities, selectable sector sizes, automatic alternate sector and track assignment, overlapped commands, data buffering, ECC, nonbuffered data transfers at disk speed, interleaved formats, backup device support, and logical or physical sector addressing.

When the systems OEM uses this last type of interface, integration problems are substantially reduced and data integrity is likewise improved. Moreover, the OEM gets the system running and into production 3 to 4 times as fast and with substantially less resources. And with the economies of scale available through volume manufacture of intelligent interfaces, the disk manufacturer can bring this type of product to the systems manufacturer very cost-effectively.

Reliability

Reliability is the key attribute of Winchester technology. Some supporting areas need to be considered as well, since Winchester has grown to be a very general term. Remember the key technology ingredients of the original Winchester technology of the IBM 3350: low mass, low force heads with contact start/stop capability, fixed media, and a sealed environment.

Today, disks and heads are permanently enclosed in most Winchester drives being manufactured. This is a characteristic vital to the improved reliability that Winchester drives provide. The most common causes of failure in pre-Winchester disk drives were improper handling of the packs and cartridges by computer operators and exposure of disk packs and cartridges to hostile environments (which includes the office environment).

The more traditional issues of electromechanical designs need evaluation. Look for fail-safe locking mechanisms for both carriages and spindles. As Winchesters find their way into more portable systems, exposure to sustained vibration and repeated shocks will put excessive demands on them.

The terms *high performance* and *stepper motor* are almost mutually exclusive. Stepper motors typically have shorter lives in the heavy duty-cycle applications of MU/MT systems, whereas voice coil motors, being designed for this kind of use, last longer. Moreover, data recovery is not as affected by temperature or vibration in drives using closed-loop servoed voice coil positioners, while open-loop systems are more sensitive to the effects of temperature change and external vibration. Sampled data systems usually take care of temperature effects, but not the instantaneous effects of shock or vibration, which may be induced by simply bumping a desktop system, or the effects of nonrepeatable spindle runout.

Vendor Viability

Analyzing a vendor's track record before committing considerable resources to that firm is an important consideration. The vendor must be able to support the prototype systems that will be put in place. Evaluate the manufacturer's technical support staff. They are the key to providing application assistance and servicing the disk drive products that are being brought to market.

The vendor must be able to support the follow-on production. Does the vendor have a record confirming its ability to ship in volume?

Does it have the financial strength to survive, as well as maintain the R&D expenditures necessary to insure continued innovation?

Cost of Ownership

Though some drives may cost more in the beginning, they may be more inherently reliable and thus require fewer service calls or repairs. Therefore, the higher initial cost may in fact result in a lower life cost of ownership.

It is predicted that improvements to disk drive technology will vastly increase disk data density (see Figure 4). By the end of the 1980s, storage of up to 60 million bits per square inch of

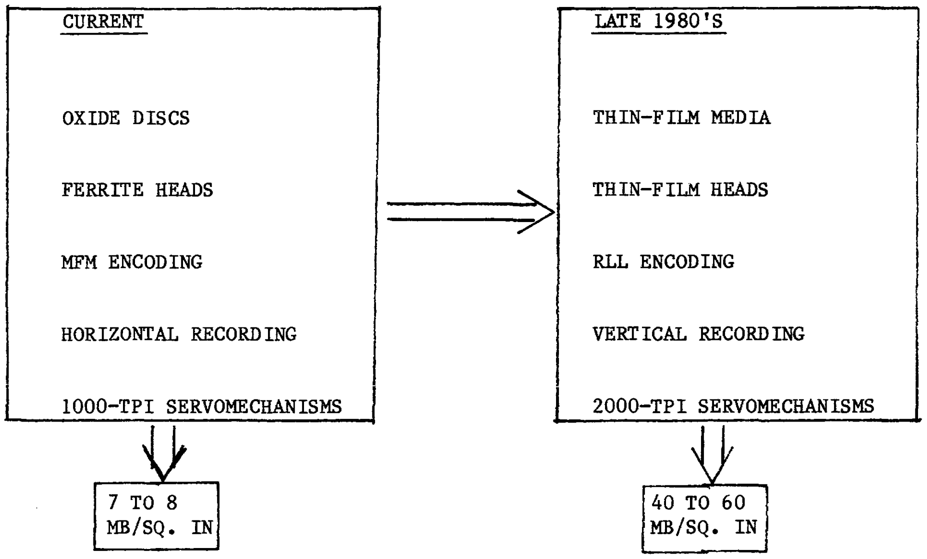


Figure 4—Disk drive capacity growth through technology

disk surface is anticipated through the use of plated or sputtered disk surfaces, new recording techniques, and thin-film heads. This potential for improvement to sealed disk drive technology makes it certain that disk drives will provide for the foreseeable future the most economical and reliable means of storing and retrieving large amounts of data.

Intel iAPX 432—VLSI building blocks for a fault-tolerant computer

by DAVE JOHNSON, DAVE BUDDE, DAVE CARSON, and CRAIG PETERSON
Intel Corporation
Aloha, Oregon

ABSTRACT

Early in 1983 two new VLSI components were added to the iAPX 432 family of components. The 43204 Bus Interface Unit (BIU) and the 43205 Memory Control Unit (MCU) extend the logical flexibility and robustness of the 432 processors into the physical implementation of 432 systems. The BIU and MCU provide a range of fault-tolerant system options. The components provide comprehensive detection facilities for processor operations as well as for the operation of buses and memories. Recovery is possible from permanent as well as transient errors. Detection and recovery are done totally in the VLSI components; there is no need for additional TTL logic or diagnostic software. This range of fault-tolerant capabilities is achieved by replication of VLSI components. VLSI replication provides software transparent migration over the full range of fault-tolerant options without any penalties for unused fault-tolerant facilities in low-end systems.

INTRODUCTION

The range and nature of computer applications is changing dramatically during the 1980s. Software, maintenance, and downtime are dominating the life-cycle costs of computer systems. At the same time, application flexibility (to grow with the user's needs and to migrate the application as new opportunities occur) is becoming a key system requirement. Matching such expanding requirements are the unfolding capabilities of VLSI technology. VLSI offers high functionality at a low cost and high reliability per function. The 432 responds to this changing environment by applying VLSI technology to significantly reduce system life-cycle costs.

The iAPX 432 was introduced in early 1980. At that time the 432 consisted of three components, a two-chip (43201, 43202) Generalized Data Processor (GDP) and a single-chip (43203) Interface Processor (IP). These processors provide an object-based architecture and modular performance growth via transparent multiprocessing. The object-based architecture of the 432 provides a robust and flexible environment for cooperating, concurrent software systems. The 432 processors use a cooperative self-dispatching mechanism to automatically share the workload between the available processors. The number of processors available in the system is transparent to software.[1,2]

The 43204 Bus Interface Unit (BIU) and the 43205 Memory Control Unit (MCU) extend the logical flexibility and robustness of the 432 processors into the physical implementation of 432 systems.[5] The BIU and MCU allow the 432 hardware to modularly and transparently expand the processing power (from 1 to 63 modules—processors or memories), bus bandwidth (from 1 to 8 backplane busses), and fault-tolerant capabilities of the system.

In the area of fault tolerance, the 432 provides a software-transparent, VLSI solution. The system may grow and migrate over a wide range of capabilities without any change to software or any cost burden for unused fault-tolerant functionality. At the low end, small low-cost systems that offer the inherent high reliability of VLSI and the ability to recover from transient errors may be configured. A range of fault-tolerant capabilities are available that increase the robustness of the error-detection and recovery facilities available in the system. At the high end, a 432 system can be configured to provide rigorous error detection over every aspect of the central system and recovery facilities for any single fault in the system. All of these capabilities are implemented in VLSI. No additional TTL logic is required, and no changes need to be made to the software system. Finally, the 432 provides an architecture that supports and encourages the development of reliable software systems. The 432 exploits VLSI technology to shift the burden of fault tolerance from software to VLSI.

The 432 provides a new level of flexibility for fault-tolerant applications that has been missing in the past. Traditionally, fault-tolerance has been considered a special and isolated set of applications. Applications were forced into an early decision either to pay for the more expensive fault-tolerant system or to choose the cheaper but less reliable standard system. The 432 eliminates this barrier by making expansion of fault-tolerant capabilities a simple and software-transparent configuration capability. The 432 is more than a fault-tolerant computer. The 432 offers cost effective solutions over a wide range of applications needs and can be modified at any time to meet the changing demands of the application.

FUNDAMENTAL PRINCIPLES

Three basic principles form the foundation for the implementation of the 432 fault-handling mechanisms.[4] First, the fault-tolerant functionality is achieved by replication of standard VLSI components, not special purpose parts. Second, the machine is partitioned into a set of confinement areas.[3] These areas form the basis for error detection and recovery. Third, only bus-oriented communication paths are used to provide system interconnection.

VLSI replication is fundamental to achieve effective use of VLSI technology. To be successful, each VLSI component must reach high-volume production. In the 432, this high-volume production is achieved by building a wide range of systems from a small set of VLSI components. The same components provide modular expansion of performance, memory storage, detection, and recovery capabilities. There are no special purpose components aimed solely at fault-tolerant functions.

The purpose of a confinement area is to limit damage from error propagation and to localize the faulty area for recovery and repair. A confinement area is defined as a unit (module or memory bus) of the system that has a limited number of tightly controlled interfaces. Detection mechanisms are placed at every interface to ensure that no inconsistent data can leave the area and corrupt other confinement areas.[3] When an error occurs in the system, it is immediately isolated to a confinement area. The error is known to be in that confinement area, and all other confinement areas are known to be error free.

By defining confinement areas, we provide a conceptual framework for mechanisms. The confinement areas also provide a conceptual view of the system under fault conditions. This clarifies the external (software) view of the hardware and significantly reduces the need for diagnostic probing as a method of fault isolation.

All communication in the 432 system is done over busses. There are no point-to-point signals or daisy-chained signals. This makes modular growth possible since no signal definition is dependent on the number of resources in the system. This approach also makes on-line repair possible. The presence or absence of any module cannot prevent communication between any other modules. The memory bus defined by the BIU and MCU provides a uniform and regularly structured communications path that supports the modular expansion of both fault-tolerant and standard system capabilities.

In the 432 there are three distinct steps in responding to an error. First the error is detected and localized to a confinement area in the system. Next, the error is reported to all of the modules in the system. This prevents the incorrect data from propagating into another confinement area and provides all of the modules with the information required to perform recovery. Finally, the faulty confinement area is isolated from the system and recovery occurs using redundant resources available in the system.

ERROR CONFINEMENT

Figure 1 shows the four types of confinement areas in a 432 system. There is a confinement area for each module and memory bus in a system. These confinement areas were chosen because modules and memory busses are the natural building blocks for 432 systems. Thus when an error is detected, it is confined to one of the system building blocks. This allows the recovery and repair strategies to be built around the replacement of system building blocks. When a module or bus has its confinement mechanisms activated, it can be viewed as a self-checking unit. The operation of a self-checking unit is designed so that no inconsistent data will be allowed to leave the unit and corrupt another confinement area. Detection mechanisms reside at every interface, and all data are checked as they flow across the interface between confinement areas.

The GDP confinement area includes the GDP and its associated BIUs plus the processor bus and support logic in the module. The only interfaces to a GDP confinement area are the memory busses. The BIUs are responsible for checking all of the information that leaves the GDP module. No information (control, address, or data) can leave a GDP confinement area without first being checked for correctness by one of the BIUs in the module. Error detection is performed by duplicating the GDP module. The duplicate module is built from identical components (GDP and BIUs) and placed in checker mode. Any disagreement is detected and signaled to the rest of the system. Thus, this duplicated module is a self-checking module.

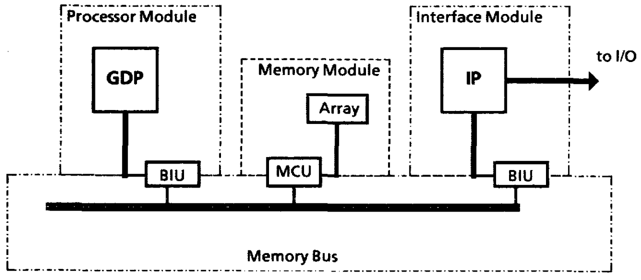


Figure 1—432 confinement areas

The IP confinement area includes the IP and its associated BIUs plus the processor bus and support logic in the module. An IP module has interfaces to the memory busses in the system, plus an interface to an external I/O subsystem. The interfaces to the memory busses are checked by the BIUs in the same manner that was described for the GDP confinement area. The IP component is responsible for checking any data that leave the confinement area via the peripheral subsystem (PS) bus. No information can leave an IP confinement area without first being checked for correctness by one of the BIUs or by the IP. The peripheral subsystem is not a confinement area. At this time the application hardware or software must apply its own detection mechanisms to this subsystem. The PS bus represents a fire wall between the central system and the I/O subsystem. The IP confinement area checks data as they leave the IP; the application HW and SW must check data that leave the I/O subsystem and enter the IP module. Error detection is performed by a duplicate checker as it was in the GDP module.

The memory confinement area includes the MCU, the RAM array, and the busses and support logic inside the module. A memory module has interfaces to two of the memory busses in the system. The MCU is responsible for checking all information that leaves the memory confinement area; no information can leave the confinement area without first being checked for correctness by the MCU. Error detection is performed by duplicating the MCU and applying an ECC code to the memory array. Thus, a self-checking memory module has two MCUs and one memory array.

Each memory-bus confinement area includes a memory bus and the interface logic residing in the BIUs and MCUs attached to the memory bus. Each memory bus has interfaces to all of the GDP and IP modules and to some of the memory modules. Every node (BIU or MCU) that is attached to this bus is responsible for checking all of the information that flows off the memory bus and into its module. No information can leave the memory bus and enter a module without first being checked for correctness by either a BIU or a MCU. Error detection is performed by two interlaced parity bits that cover the control and address/data lines. Error detection for the arbitration lines is achieved by duplication. A timeout is used to detect protocol violations on the bus.

An example processor memory operation may help to clarify the operation of the confinement areas. (This example is shown graphically in Figure 2.) Assume a GDP makes a read request to a memory location. That request will be mapped through the BIU on the addressed memory bus. As the information flows onto the memory bus it will be checked by the BIU. If there was any failure in the GDP confinement area (GDP, processor bus, BIUs, etc.) it will be detected at this time. The information flows across the memory bus and into the addressed memory module. Before the information is accepted by the module, the MCU checks it for correctness. If a failure is detected it is confined to the memory bus because the information was valid when it left the GDP con-

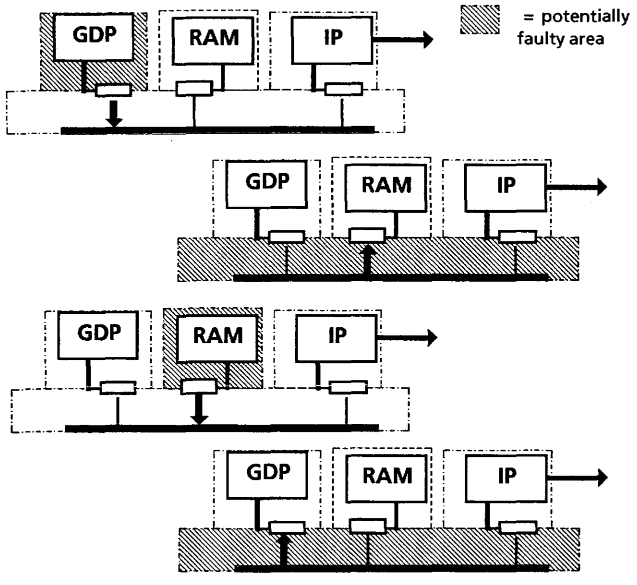


Figure 2—Confinement area operation

finement area. The MCU performs the memory operation and returns data onto the memory bus. As data flows onto the bus it is checked for correctness by the MCU. As the data flows into the GDP module from the memory bus it is checked for correctness by the BIU before being used by the GDP module.

The confinement area interfaces provide very tight error control and isolate the failure to one of the building blocks present in the system. These confinement areas were formed by applying five different detection mechanisms: duplication, parity, hamming error-correction codes, timeouts, and loop back checks (used to detect errors in the TTL bus drivers). The only remaining question concerns checking the detection mechanisms. Some of the detection mechanisms are self-checking (the detection circuits are checked as part of normal operation). Those circuits that are not self-checking can be exercised during normal system operation to flush out any latent faults.

REPORTING

Immediately upon detecting an error, a message is broadcast to all the nodes in the system. This error-report message identifies the faulty confinement area, the type of error that occurred, and whether the error is permanent or transient. There are two reasons for sending this error report. First, it informs the rest of the system that an error has occurred, which prevents other confinement areas from using the inconsistent data. Second, it provides the necessary information for system recovery. After recovery, the error message is recorded in a log register in every node in the system. This log is available to software and is useful in monitoring the health of the system.

The error messages are broadcast over a set of serial busses that are totally independent from the busses used during normal operation. This error-reporting network is fully fault tolerant. No single failure in the error-reporting network can prevent the correct and timely reporting of an error in the system. This network of serial busses follows exactly the same topology as the busses used for normal operation. A failure on one of these busses is limited to one of the confinement areas discussed earlier. The failure of one error-reporting bus does not compromise the fault-tolerant capabilities of the system. Other error-reporting busses (in the other confinement areas) take over its reporting responsibilities. The error-reporting circuitry may be tested during normal operation to uncover any latent faults.

RECOVERY

The recovery process begins after an error-report message has been broadcast around the system. Recovery is a distributed operation on the 432. Each node in the system reads the error-report message and decides what recovery action needs to be taken.

For recovery to be successful, there must be redundant resources available in the system. There are five redundancy mechanisms in the 432. Retry and single-bit correcting ECC codes are used for recovery from transient errors. Shadowed modules, backup busses, and spare memory bits are used for recovery from permanent errors. These redundant resources cover the entire system and allow recovery from any detected error. The presence of redundant resources has no impact on system performance.

For transient errors: Each BIU maintains an internal buffer that allows outstanding processor requests to be retried if a transient error occurs. A single-bit correcting ECC code is applied to each word in the memory arrays. Although this provides redundancy for both permanent and transient errors, its primary purpose is to correct soft errors that occur in dynamic RAMs.

For permanent errors: Every self-checking module in the system may be paired with another self-checking module of the same type. This pair of self-checking modules operates in lock step and provides a complete and current backup for all of the state information in the module. This mechanism is known as module shadowing because the shadow is ready for immediate recovery should the primary fail, or vise versa. A fault-tolerant module is also called a quad modular redundant (QMR) module because the 432 VLSI components are replicated four times (two self-checking modules, with a master and checker in each module).

Each memory bus in the system may be paired with another memory bus. During normal operation the busses run independently; both contribute to the total bandwidth available in the system. However, if one bus fails, the other bus is capable of handling the bus requests that normally would have been handled by the failed bus. If one bit in the array fails, the spare bit can be switched in to replace the failed bit.

For transient errors, all of the outstanding accesses are retried and the MCUs return corrected data if there are any single-bit errors in the memory arrays.

For permanent errors, the redundant resource is switched in to replace the failed unit. This switch is done on a node-by-

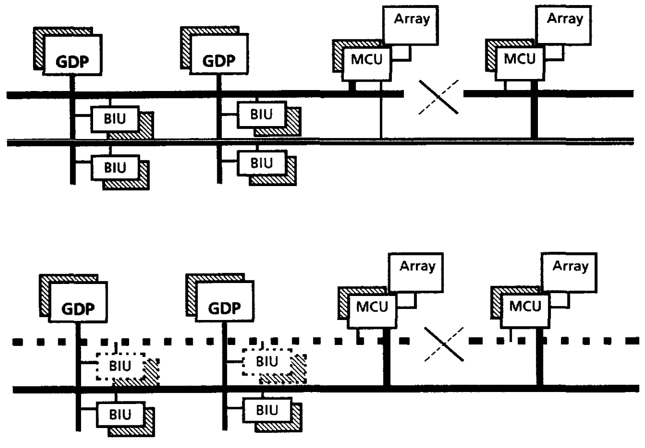


Figure 3—Bus reconfiguration

node basis; there is no centralized element that controls the switch. Each node knows which module or memory bus it is backing up (shadowing). If the error report identifies its partner as the faulty unit, then the node becomes active and takes over operation for the faulty unit. After the resource switch is complete, all of the outstanding memory accesses are automatically retried. This allows operation to resume at a point before the failure corrupted data.

These reconfiguration and recovery actions are performed by the hardware without any software intervention. After recovery is complete, the hardware informs the system software of the error and subsequent recovery actions. System software now makes policy decisions regarding the optimum system configuration given the resources remaining in the system. The software may choose to maintain full capabilities of the system and switch in a spare resource, or maintain fault tolerance and degrade performance by switching out the unit that no longer has a shadow, or maintain performance and run with an increased probability of failure by keeping the shadow unit in operation without bringing a spare on line. All of these configuration options are under software control and require no manual intervention or hardware changes. These policy decisions are carried out while normal system operation continues. Figures 3 and 4 show two examples of recovery operations.

SUMMARY

The 432 fault-tolerant mechanisms are designed to provide a flexible and complete solution to the problems of fault-tolerant hardware. For basic systems, a user may decide to use only a few detection mechanisms and to provide recovery for only transient errors (no checkers for error detection or shadows

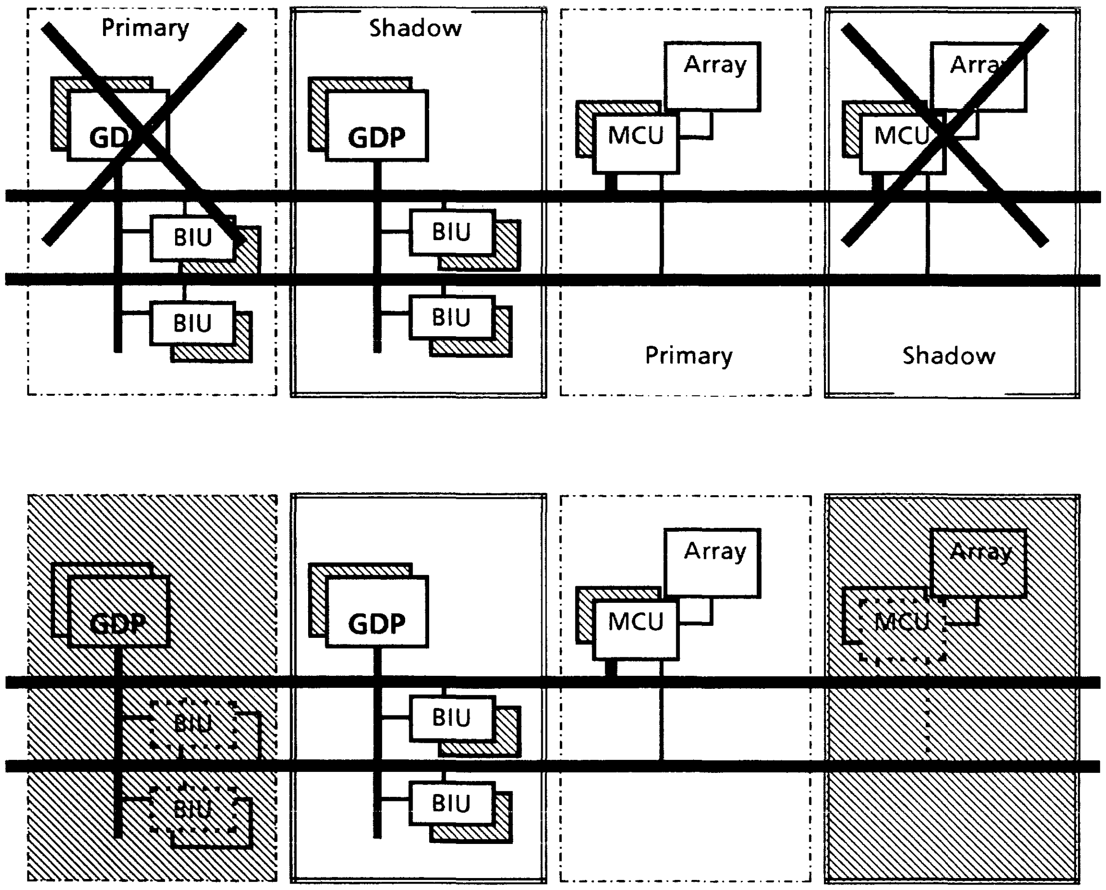


Figure 4—Module reconfiguration

for recovery). This functionality comes without extra cost in the VLSI interconnect system. To reduce maintenance costs and increase system availability, a system may use all of the detection mechanisms but may not add any extra recovery capability (add checker modules, but not form shadow pairs from the self-checking modules). Fully fault-tolerant systems (continuous operation in the presence of any single failure) are available to the user by adding the extra recovery capabilities. This expansion of fault-tolerant capabilities is accomplished solely by replicating VLSI components, not by the addition of new component types.

None of the fault-tolerant mechanisms reduce system performance. Systems that do not require the highest level of fault tolerance are not penalized in any way (either in cost, size, or performance) for the unused fault-tolerant capabilities. Increased levels of fault tolerance are achieved by replicating the 432 VLSI components. The hardware fault tolerance in the 432 is transparent to application software. The system's fault-tolerant capabilities may be changed without any changes to the application software system.

ACKNOWLEDGMENTS

The 432 fault-tolerant design would not have been possible without the support and design refinement by many talented people within Intel's Special Systems Operation. In particular, we would like to thank Tony Cornish (who was a co-architect of this work) and Bill Corwin for their important contributions to the design work.

REFERENCES

1. Intel Corporation. *Intel 432 System Summary: Manager's Perspective.* Santa Clara: Intel, 1982. Order number 171867.
2. Intel Corporation, *iAPX 432 GDP Architecture Reference Manual.* Santa Clara: Intel, 1982. Order number 171860.
3. Denning, Peter J. "Fault Tolerant Operating Systems." *ACM Computing Surveys,* 8 (1976), pp. 359–389.
4. Siewiorek, D., and R. Swartz. *The Theory and Practice of Reliable System Design.* Digital Press, 1982.
5. Intel Corporation. *iAPX432 Interconnect Architecture Reference Manual.* Santa Clara: Intel, 1983. Order number 172487.

Performance evaluation of the MP/C

by BRUCE W. ARDEN
Princeton University
Princeton, New Jersey
and
RAN GINOSAR
Bell Laboratories
Murray Hill, New Jersey

ABSTRACT

The performance of the MP/C, a multiprocessor/multicomputer architecture, is evaluated by means of deterministic and probabilistic techniques and simulations. The MP/C is examined in two different applications, as a tree machine and as a multiprogramming general-purpose system. The second application of the MP/C is compared with a star architecture. The evaluation suggests that the MP/C is very effective for certain special applications, such as tree algorithms, and worst-case analysis shows that its performance is acceptable even for general purpose computations.

A. INTRODUCTION

The architecture of the dynamically reconfigurable MP/C has been described in detail.[1] It consists of a linear bus with multiple processors and memory modules. Figure 1 shows a simplified MP/C system and a few possible configurations. Switches are positioned on the bus between every pair of processors, and the bus can be partitioned into any configuration of segments. On each such segment, only one processor is active, and it accesses all the memory modules on its segment. Open switches provide effective isolation of segments and guarantee full mutual exclusion of active processors. A processor may activate another processor in its segment by opening the switch immediately to the left of the activated processor. The memory space of the activator is split, and part of it is exclusively assigned to the activated processor. The switch control logic supervises this reconfiguration operation. Such an activation is, in effect, a hardware implementation of a *process-fork* operation. The converse operation, *process-join,* is achieved by deactivating a processor, closing the switch to its left, and thus reattaching its memory to the neighboring active processor.

The applicability of most indirect performance evaluation tools, such as mathematical analysis and simulation, to an architecture as complex as the MP/C is rather limited. The many unconventional attributes of the MP/C, especially its reconfigurability, are beyond the capabilities of most analytic techniques. Simulations also involve a lot of approximations, which reduce the usefulness of their results. The desirable techniques would be an emulation system (which inputs, executes and measures real MP/C code) and a measurement on a full-scale hardware implementation. Both techniques are currently being pursued.

In this report we describe some preliminary performance evaluation studies. Two situations are considered. In the first we examine the MP/C in the context of tree-structured computations. As is emphasized elsewhere,[1] it is expected that this application is the most appropriate one for the MP/C. The second part investigates the performance of the MP/C in running a general-purpose, multiprogramming operating system

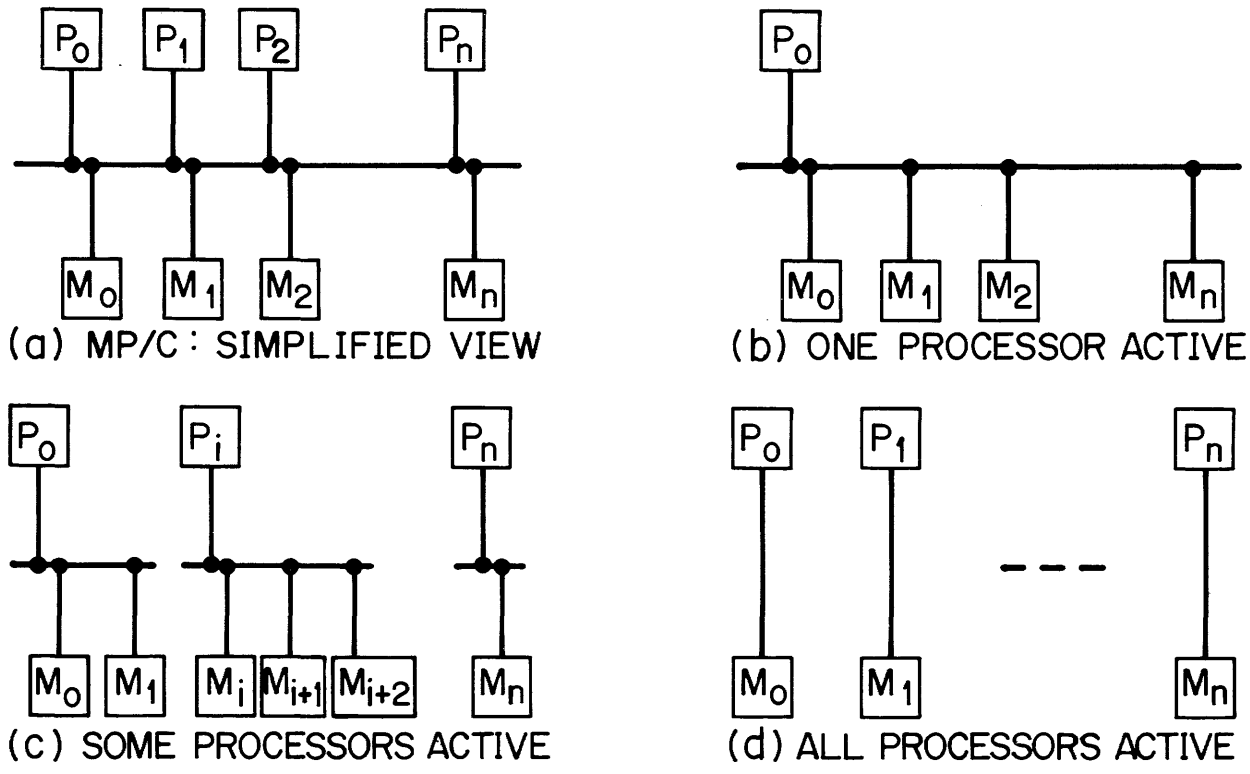


Figure 1—Simplified diagram of an MP/C with possible reconfigurations

and many independent tasks. Although this is not the best application for the MP/C, the evaluation serves two purposes. First, not every algorithm is tree structured, and it is economical to have a machine that is efficiently applicable to more than one computational structure (preferably, it should provide all the computational needs of its owner relatively efficiently). Second, it is a worst-case study: given an unfavorable computational structure, what is the lower bound on the performance of the MP/C?

The next section analyzes the performance of tree-structured algorithms. Section C introduces multiprogramming applications on the MP/C. Section D employs a deterministic model to arrive at some preliminary conclusions. Sections E through I describe the analytic models and their results. We focus on techniques for modeling preemption, explain the MP/C and star models, present the results and their interpretation, and describe an alternative MP/C model. The simulation study is described in Sections J through L.

B. TREE-STRUCTURED ALGORITHMS

Since only one processor is active in each MP/C partition, the throughput (in terms of the collective instruction execution rate of all the processors) varies, and depends on the number (or, equivalently, size) of the partitions. Since the partitioning is really an image of the process structure of the running algorithm, the amount of parallelism in the MP/C generally reflects closely the level of inherent parallelism in the algorithms. For example, consider a divide and conquer algorithm.[1] Five phases can be identified:

1. The input phase, in which the data and programs are loaded into the MP/C. This phase probably takes close to the same amount of time on the MP/C or any other organization, so it can be dropped from consideration.
2. The forking phase, in which a series of activations follows top-down scanning of a binary tree. In each step the number of active processors is doubled. In the merge-sort example, there is only a limited amount of computation in each step, provided the data and program are loaded effectively. In that case, this phase takes $O(\log n)$ time, with n processors. In some algorithms, like parallel search, all leaf processes are spawned by a single parent in the process-forking tree, rather than recursively spawned according to a binary tree. In those cases, the successive activation may be replaced with the single step PARALLEL-ACTIVATE operation, in which all processors are activated simultaneously.
3. In the totally parallel phase all processors are active, and throughput is maximal. When the number of data elements involved is larger than the number of processors (usually the case), each processor executes a sequential algorithm on its subset of the data during this phase.[1] In the merge-sort example with N data elements and n processors this phase takes $O((N/n)\log(N/n))$, which is close to a linear speedup (of this phase only). Similar speedup is observed for other examples.
4. The joining phase, which is the converse of phase 2. In merge-sort, at each successive step the merge operation runs twice as long while employing only half the number of processors, and takes time $O(N)$, usually longer than the previous phase. On the other hand, search and retrieval algorithms do not require heavy computation at this phase (unless a substantial amount of data is retrieved), and some other algorithms require only $O(\log n)$ time.
5. The output phase, which has a structure similar to the input.

It is clear that maximum throughput (and linear speedup) is achieved only in phase 3. Thus, the throughput depends on the ratio of the time complexity of phase 3 to that of phases 2 and 4. Similar behavior is demonstrated by other classes of algorithms.

Most algorithms execute on the MP/C with asymptotic time complexities similar to what they would have on other parallel-processing architectures. However, constant factors may differ greatly, and indeed might make all the difference. The most important advantage of the MP/C in that regard is its ability to switch memory blocks. Where multicomputers share data by moving them around in messages, the MP/C avoids most such data movements. For applications like merge-sort, where all the data have to be moved in a tree multicomputer, this moving takes $O(N)$ time for N data elements. In addition, the overhead associated with message passing (packing, serializing, unpacking, etc.) is saved. On the other hand, shared-memory multiprocessors also avoid message passing, but the contention associated with the shared resources limits the effective number of processors, as we have said.

In summary, the performance advantage of the MP/C is characterized both by an improvement of constant factors and by an ability to support many processors. However, it is difficult to prove constant-factor improvement without actually implementing the proposed architecture in hardware and comparing its measured performance to that of a hardware implementation of other architectures.

C. THE MP/C AS A GENERAL-PURPOSE, MULTIPROGRAMMING COMPUTER

In considering the utility of the MP/C for running a general-purpose operating system, and its associated independent user tasks, a different approach is taken. As long as all processors run user tasks, and as long as those tasks require only computing, the throughput is maximal. When a task, running on P_i, requires system services, or when it completes and has to be swapped, P_0 has to access M_i by preempting all the intermediate processors. We model the effect of preemption through the multiclass network of queues. Each running process is characterized by the ratio of pure computing time to I/O time it requires, and this ratio can vary from one process to another.

Utilization of memory is also an important performance factor, especially in a microprocessor-based MP/C where a single memory block may be more expensive than its associated processor. Although it seems that it would be possible to effectively employ more processors on the same amount of memory in the MP/C than in strict multiprocessors, the fact that common programs cannot be shared and each running processor must store its own copy of the program reduces the potential memory utilization. This effect, however, should be weighed against the relative benefits of the MP/C.

Three methods were employed: deterministic evaluation, analytic modeling using queueing networks, and simulation. The purpose of the evaluation study was to find lower bounds on the performance. That is, we model the MP/C under the least favorable conditions, and try to evaluate its performance.

Since absolute performance figures are generally not meaningful, this is mainly a comparative evaluation. We first identify an unique feature of the MP/C (certain preemptions), and compare it to another architecture (the star) that differs from the MP/C only in that feature. Thus, the comparison isolates the effects of that feature.

There is no single architecture which can efficiently execute the full range of concurrent computations, but the reconfigurable MP/C is intended to be usable over a fairly wide spectrum. The hierarchical nature of the MP/C makes it a candidate for the execution of a general-purpose, process-structured operating system (e.g., UNIX). In such an environment, user processes and most of the system processes are spawned by the kernel process, and constitute a shallow but wide tree. That is, most processes are direct children of the kernel. On the MP/C, such a system would run the kernel on P_0, and the user processes on the other processors. For simplicity, assume that each user is allocated a single processor and a single memory module, and that no inter-user communications are needed. Also, we ignore here the possibilities of running a user process on P_0 (in addition to the kernel), of running the kernel on another processor, and of using a distributed kernel.

It is known that the speedup of such a multiple-processor organization is generally sublinear. Among the factors that usually contribute to that are memory contention, system contention, and synchronization between processes.[3, 11, 13, 21]

In addition, another source of degradation is unique to the MP/C. Whenever a process that runs on some processor P_i has to communicate with the kernel, the bus segment starting at the kernel processor and up to P_i must be available, that is, all the processors between the P_j, $0 < j \leq i$ must be preempted. The only active processor in this segment would be P_0, running the kernel. Such communications are required when a process initiates I/O operations, when it receives resulting data, when a page fault occurs, when the process initiates any type of system service request, and when a process terminates and the next process has to be loaded. The evaluation of the effect of this requirement is the subject of this paper.

D. DETERMINISTIC MODELING

Consider a deterministic model for the star. The kernel executes on P_0, and the P_i, $i = 1, \ldots, N$ are user processors. Each user processor is allocated a single user job, which goes

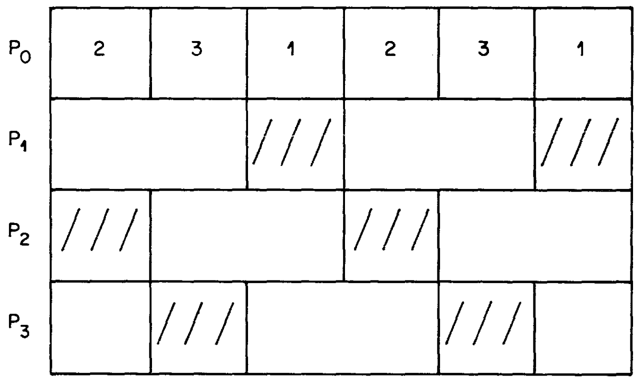


Figure 2—Deterministic scheduling of a star

through the following fixed cycle. It first executes on the user processor for β units of time, and then requires service from the kernel for one unit of time. Since there are N users, and each requires one time unit of service by the kernel, the cycle time of each job is

$$C_{\text{star}} = \max\{\beta + 1, N\}.$$

When $N < \beta + 1$, the kernel processor P_0 is underutilized. When $N > \beta + 1$, interference between jobs (congestion at the kernel) increases cycle times beyond the necessary minimum $\beta + 1$. Hence, we define a *balanced star system* as one with $N = \lceil \beta + 1 \rceil$. Note that $\beta + 1$ can also be interpreted as *effective parallelism*. Given N, for $\beta << 1$ the kernel is highly congested and the whole system behaves effectively as a uniprocessor. On the other hand, for $\beta > N$ the system shows linear speedup without any interprocessor interference. The utilization of the kernel processor is

$$\rho_{0,\text{star}} = \frac{N}{C}\bigg|_{N \geq \beta+1} = 1.$$

The utilization of each user processor is

$$\rho_{i,\text{star}} = \frac{\beta}{C}\bigg|_{N \leq \beta+1} = \frac{\beta}{\beta + 1}, \; i = 1, \ldots, N.$$

In Figure 2 we show an example of a Gantt chart [C1, CD] of a balanced system with $N = 3$ and $\beta = 2$. Since the system is balanced, there is no queueing at the kernel, and the kernel is fully utilized. This is a snapshot of 6 time units during steady state operation. The tasks at P_0 are labeled by the user processor which they serve. The slanted areas designate idle periods at the user processors.

Now consider a similar model for the MP/C. The difference is that when the kernel serves the job of P_i, all user processors between P_i and the kernel must be preempted. This is represented by a task that executes concurrently on all of them. In Figure 3, such tasks appear as the vertical columns. Note that only the topmost processor in a vertical column is considered active for the purpose of computing utilization. As before, each job requires 1 time unit of service from the kernel, and

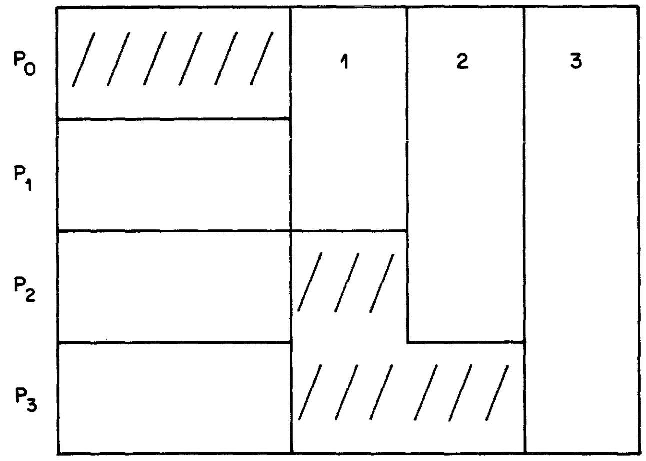


Figure 3—Deterministic scheduling, MP/C with fixed β

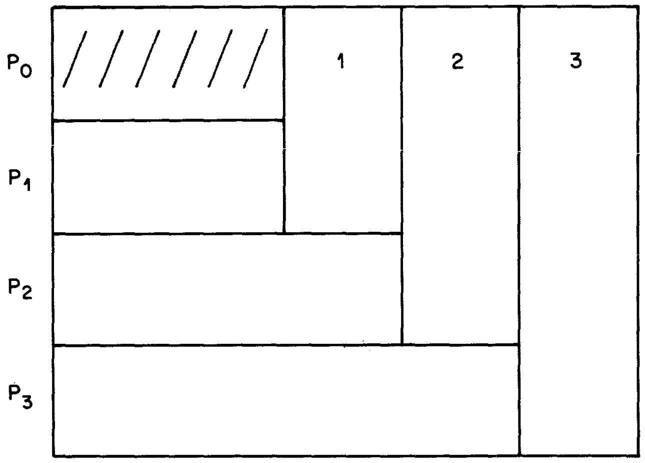


Figure 4—Deterministic scheduling, MP/C with variable β

β units of computation on the user processor. The kernel processor utilization is

$$\rho_{0,\mathrm{MP/C}} = N/(\beta + N),$$

and the utilization of each user processor is

$$\rho_{i,\mathrm{MP/C}} = \beta/(\beta + N),\ i = 1, \ldots, N.$$

Unlike the star, MP/C has no value of N for which any processor is fully utilized, and interference always exists. Instead, we say that the MP/C is a *balanced system* when overall utilization is maximized. One obvious case is when $\beta \rightarrow \infty$, with N fixed. The other case is derived as follows. Given β, and treating N as a continuous variable,

$$\lim_{N\to\infty} \rho_{\mathrm{MP/C}} = \lim_{N\to\infty} [(1/(N+1))(\rho_{0,\mathrm{MP/C}} + N\rho_{i,\mathrm{MP/C}})]$$
$$= \lim_{N\to\infty} [N(1+\beta)/(N+1)(\beta+N)] = 0.$$

Also, for $N = 0$, $\rho_{\mathrm{MP/C}} = 0$. Assuming $\rho_{\mathrm{MP/C}} > 0$ in the open interval $N \in (0, \infty)$, there exists a maximum for $\rho_{\mathrm{MP/C}}$ in that interval. We equate to zero the first derivative of $\rho_{\mathrm{MP/C}}$ with respect to N:

$$(\partial/\partial N)\ \rho_{\mathrm{MP/C}} = (\partial/\partial N)\ [(1/(N+1))(\rho_{0,\mathrm{MP/C}} + N\rho_{i,\mathrm{MP/C}})]$$
$$= (\partial/\partial N)\ [N(1+\beta)/(N+1)(\beta+N)]$$
$$= \frac{(1+\beta)(N^2 + N(1+\beta) + \beta) - N(1+\beta)(2N+\beta+1)}{[N^2 + N(1+\beta) + \beta]^2} = 0.$$

We get

$$N^2 + N(1+\beta) + \beta = 2N^2 + N(1+\beta)$$

or

$$N = \lceil \sqrt{\beta} \rceil.$$

Since there is only one value of N in the interval $(0, \infty)$ for which the first derivative is zero, and since $\rho_{\mathrm{MP/C}}$ achieves its minimum of zero in both ends, this is a maximum, and we need not consider the second derivative.

In other words, under these conditions the MP/C seems to support effectively less parallelism than the star. This relative advantage of the star is achieved at the cost of more elaborate (and more expensive) communication paths.

The asymmetry in the MP/C suggests that the MP/C can be better utilized if the load is not the same for all processors. In Figure 4, we take advantage of some of the idle time of the user processors, by introducing *nonuniform* loads of β, β + 1, and β + 2 to P_1, P_2, and P_3 respectively. The kernel utilization is the same as above. The total (or average) utilization of the user processors is

$$\bar{\rho}_{Users,\mathrm{MP/C}} = \frac{1}{N}\sum_{i=1}^{N} \rho_i = \frac{1}{N}\sum_{i=1}^{N} \frac{\beta + i - 1}{\beta + N} = \frac{2\beta + N - 1}{2(\beta + N)}.$$

Using the same procedure as above, we get

$$N = 1 + \lceil \sqrt{2(\beta+1)} \rceil$$

as the value of N maximizes utilization. Although still lower than $N = \beta + 1$ of a balanced star, it is better than $N = \lceil \sqrt{\beta} \rceil$ of the uniform-workload MP/C. In any case, the observation earlier made about effective parallelism is applicable here as well; given N, low values of β imply effectively sequential processing, whereas $\beta >> N$ assures virtually interference-free operation. The lesson of this discussion is that it is meaningful and practical to consider the MP/C only under high β workload.

E. ANALYTIC MODELING: MODELING PREEMPTION

We present an approximate modeling of preemption that produces results for the steady-state analysis of preemption in a network of queues. It is based on techniques employed for the modeling of message passing in multicomputer networks.[1] This technique may also be applicable to other queueing systems.

Assume that a job j_1, which must execute for t_1 units of time, starts execution at service center c_1. Job j_2 arrives at that center τ units of time later and preempts j_1 for time t_2, after which job j_1 resumes execution.

A variation of this situation, which is more interesting in the case of the MP/C, occurs when j_1 (running on c_1) is preempted for t_2 units of time, but c_1 is left idle and no other job executes on it during the preemption period t_2. This is the case when some resource that is allocated to c_1 is temporarily needed elsewhere (bus and/or memory in the case of MP/C). We model this idle preemption by a job that executes on the preempted processor for the same amount of time, t_2.

An accurate modeling of this sequence would be as follows. First, jobs j_1 and j_2 execute on centers c_1 and c_2, respectively. (See Figure 5a.) Then, j_2 spawns a new process, $\bar{j}_2$, which preempts j_1 and runs on c_1 (Figure 5b). Finally, $\bar{j}_2$ releases c_1 and terminates, and j_1 resumes, as in Figure 5a again. However, the analytical tools do not allow the spawning of processes as part of internal transitions in queueing networks.[7] Hence, another modeling technique is necessary.

In order to overcome this difficulty, we make use of the following observations. The details of scheduling are quite important for the outcome of the computation, but not all of the details are relevant for the steady-state performance analysis. In particular, it is important to know t_1 and t_2, but the relative order of execution, and the length of τ (the time from execution of j_1 and j_2 on c_1 as a sequence (j_1 executes first to completion and only then does $\bar{j}_2$ execute), rather than having $\bar{j}_2$ preempt j_1. (Note that in some cases this approach may not be valid.) Thus, the spawned job in the example above, $\bar{j}_2$, would not generally execute concurrently with its originator j_2, but rather at some later time, when c_1 can serve it. Since it is no longer mandatory that j_2 and $\bar{j}_2$ execute simultaneously, the solution to the spawning problem is simple: when j_2 finishes execution on c_2, it changes its class from a *productive* one to a *non productive*, moves from c_2 to c_1, and executes there for time t_2, that is; the preemption time. This execution does not model real computation, but rather the time that c_1 is preempted and is not available for productive work. Following that, the job may return to c_2.

c1
j1
c2
j2
j1
j̄2
j2
Figure 5—(a) Jobs j_1 and j_2 execute on center c_1 and c_2.
(b) Job j_2 preempts j_1.

On the other hand, note that this technique also introduces two modeling errors. The fact that job j_2 has to spend t_2 in c_1 (in the non productive state) may delay the next activity of j_2 (for time t_2), and this might affect the total system performance, as it does in the case of the MP/C (see below). The second error is due to the successive possession of resources, rather than concurrent, as is implied by preemption. That is, in the model j_2 possesses c_2 and c_1 in sequence, rather than simultaneously. The waiting time to possess two resources concurrently may be longer than the sum of the waiting times to possess each independently. Thus, the first error may result in underestimation of the performance, whereas the second one probably has the opposite effect.

Among the available scheduling disciplines that can be modeled analytically, the last-come-first-served-preemptive-resume (LCFS-PR) discipline is the most suitable one.[7] (Other disciplines, PS and in certain cases FCFS, may render the same results.)

Both productive processing and preemption time are thus modeled by active computation, although by different sets of classes. This separation of classes allows the evaluation of performance from two different viewpoints. For instance, the utilization in the productive classes reflects the real work done by the processor, for productive computation. On the other hand, tht total utilization (productive and nonproductive) reflects that portion of time in which the processor is busy. Thus, if the productive utilization is 0.5 and the total utilization is 0.8, it can be concluded that although the processor is productively active only about half the time, no more than 0.2 additional utilization can ever be expected. Which of the two criteria is more interesting depends on the application.

F. ANALYTIC MODELING: THE MP/C MODEL

In the MP/C employed as a general-purpose computer, preemption is used to allow direct communications between the leftmost processor P_0 and some memory module M_i. All processors $P_j, 0 < j \leq i$, must be preempted. The lost time is modeled by a job that starts at P_{i-1} and progresses leftward until it runs on P_0. (See Figure 6). After finally finishing at P_0, this job returns to P_i for the next cycle of computation.

All service times are drawn from exponential distributions. Two classes are associated with each processor P_i: the productive class i, in which a job runs on P_i with expected service

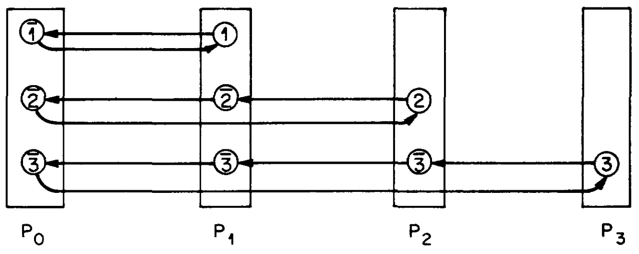


Figure 6—The MP/C model

time of one unit, and the nonproductive class $\bar{i}$ designating this job when it goes through $P_{i-1}, \ldots, P_0$. The average service time on each of those processors is α, the time it takes for P_0 to communicate with M_i.† The transition probability matrix is thus trivial and contains only zero-one elements.

Note that an approximation error is introduced here. Suppose each single MP/C primitive operation (e.g., PREEMPT) takes k units of time. In each k units a processor can issue on PREEMPT instruction, which preempts only the next active processor down the line.[1] Thus, the preemption of all processors between P_0 and P_i is a multi/step operation. Similarly, at each step the kernel can resume the operation of only one other processor. Hence, reactivating all the preempted processors is a multi/step operation too. In other words, instead of all these processors being preempted for α units of time, P_i is preempted for α units, P_{i-1} is preempted for $\alpha + 2k$ units, $\ldots, P_j$ is preempted for $\alpha + (i-j) \times 2k$ units of time, $0 < j \leq i$. However, as a first-order approximation, we assume that $2ki$ units of time are negligible compared to α units of time, and each processor is preempted for the same period. It may be possible to accommodate this detail later by adjusting the mean service times appropriately.

As explained above, many types of user-process-to-kernel communications may take place. In order to isolate the behavior of the MP/C from peculiarities of specific benchmarks and workloads, a simple execution profile was chosen. Each processor is assigned a single job. Each job does one of two things: it computes on its processor, or it requires service from the kernel. The first task is called *computation,* and the service time for it is a random variable drawn from exponential distribution with mean of 1 unit. The second task is called I/O. It is exponentially distributed, and requires variable mean service time α. On the MP/C, α represents the length of time for which the kernel has exclusive access to the memory of the current processor. High values of α represent I/O-bound jobs. By varying the ratio of the I/O time to computation time, $\alpha/1$, from 0 to ∞, we are able to study the performance of the MP/C across the full range of workload. For two reasons, this approach is superior to picking up some small set of workload values for which the MP/C performance is maximal. First, we have been able to uncover unexpected and rather surprising behavior at different workloads. Second, the MP/C is intended for general purpose applications, in which any type of workload can be expected. For instance, it is conceivable that jobs that have α values of 100 and 0.01 will execute on the same machine concurrently. Recall however that it was argued in Section D that the reasonable workload range is $\alpha < 1/N$ (corresponding to $\beta > N$).

At first, we model the case where all the jobs on the same MP/C have the same α value. Another case is when different jobs have different α values. It is reasonable to expect that, in the I/O phase, the farther the processor is away from P_0, the higher the incurred penalty for the whole system, because more processors must be preempted. Thus, it may be advantageous to assign the jobs with higher degress of α to processors that are closer to P_0. We try to verify this assumption, and

†This is the reverse of the notation of Section D: $\alpha = 1/\beta$.

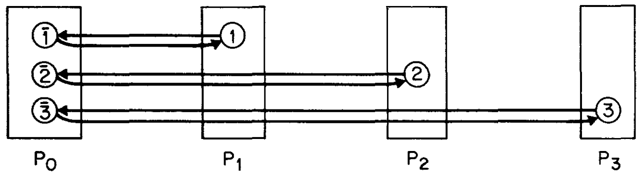


Figure 7—The star model

study the effect of optimal and nonoptimal assignment of processes.

G. ANALYTIC MODELING: THE STAR MODEL

The star is used as a baseline to which the performance of the MP/C is compared. It is chosen because in this case (the MP/C as a general-purpose machine) the star differs from the MP/C in exactly that characteristic which is under study, interprocess communications. (Note, however, that, unlike the MP/C, the star is limited to a relatively small number of processors, owing to the high degree of connectivity required of its central node.) In a star, the kernel is run on the central node, and each processor has a direct link to the kernel processor. Figure 7 shows the star model, drawn as a linear structure, to emphasize the structure common to the MP/C. As in the MP/C, each job runs on a processor for one unit of time (average). Unlike an MP/C job, it then goes directly to the center (P_0) and executes for α units of time, after which it returns to its processor.

H. ANALYTIC MODELING RESULTS

H.1. Processor Utilization

We first consider the utilization of user processors. We used a queueing network solver program, PNET, developed by Bruell and Balbo.[5,6] It is a PASCAL program, running on VAX UNIX, and (on our configuration) it can handle only up to 8-processor MP/C models. Figure 8 shows the productive utilization of the user processors, P_1 through P_7, in an 8-processor MP/C (one kernel, seven users), plotted versus the workload parameter α. Recall that α models the ratio of preemption time to compute time. These curves justify the claim that the important range of workload is $\alpha < 1$. The curves show that, in the model, the farther a processor is from P_0, the lower its productive utilization is. Although this is a straightforward result for the model, we show in Section L that this is not exactly the case in a real MP/C.

We choose to observe productive utilization, rather than total utilization. Figure 9 shows an example of total utilization of three user processors. Some of the processors are heavily utilized even for high values of α, but this high utilization only reflects the fact that they are preempted for long durations, not that they are productive. Total utilization in this case obscures the results and makes it noncomparable to the star, where there is only productive utilization. Hence, in the following "utilization" refers only to productive utilization.

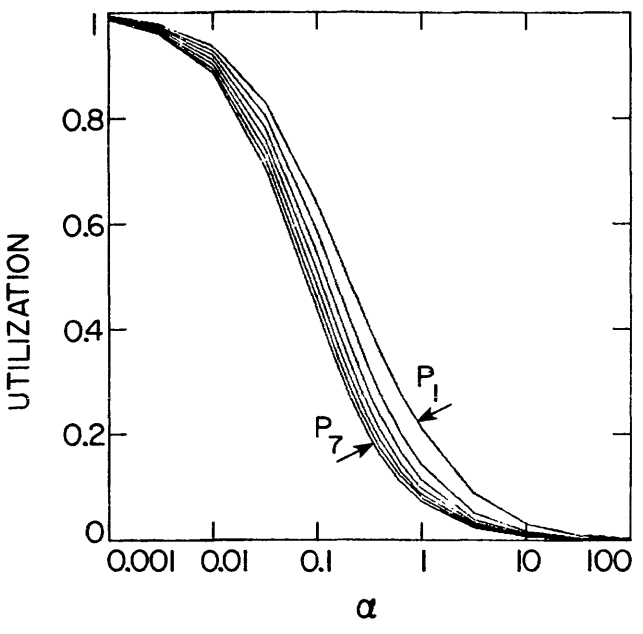


Figure 8—MP/C productive utilization

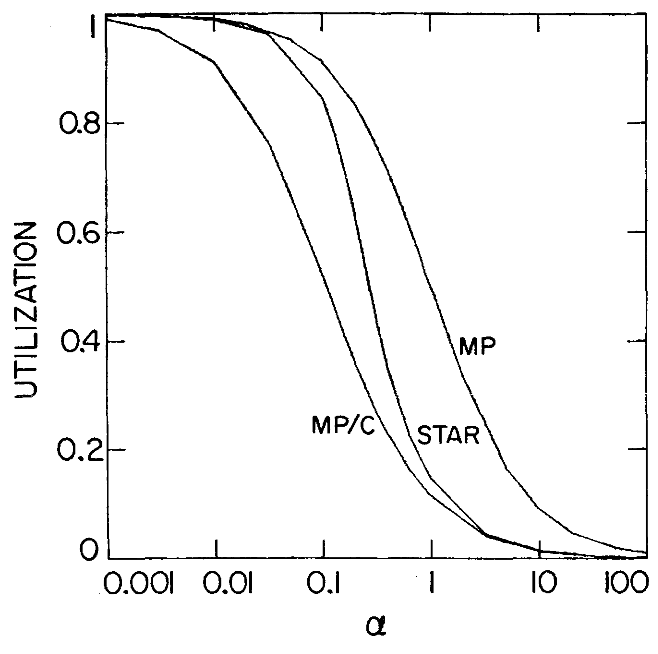


Figure 10—Average utilization: MP/C, star, MP

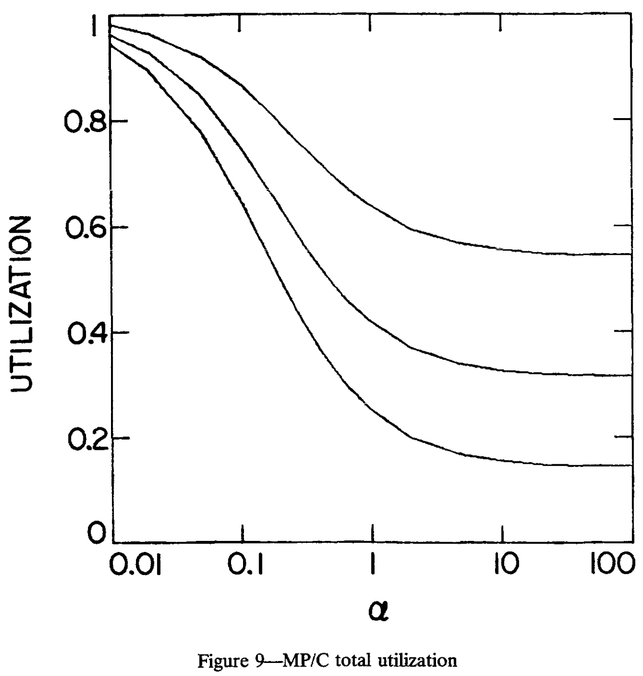


Figure 9—MP/C total utilization

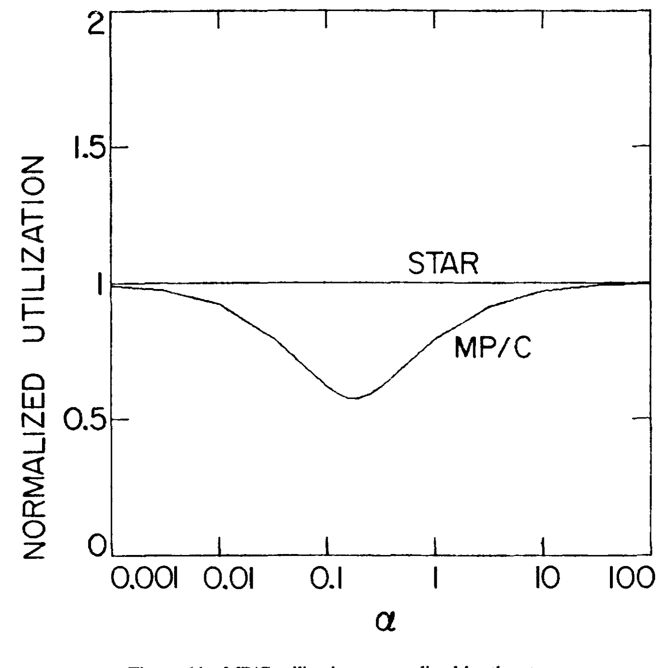


Figure 11—MP/C utilization, normalized by the star

The information from Figure 8 is condensed into a single curve of average utilization in Figure 10. It also contains the average utilization curve for the 8-processor star (one center, seven users). Since all processors in the star are equal, the average is also the curve of each of them. For reference, a third curve (marked |MP|) shows the utilization of an "ideal" multiprocessor. By ideal we mean that no interference between processors exists, and this can be viewed as modeling seven processors, each having its own I/O server. The utilization is defined there as $1/(1 + \alpha)$, because 1 is the computation time and $1 + \alpha$ is the total time.

Figure 11 shows the (productive) utilization of the MP/C normalized by that of the star. Figure 12 shows all the curves for N, the number of user processors, $2 \leqslant N \leqslant 7$. Note the following six phenomena:

1. The curves show a dip in midrange. In this area, the performance is very sensitive to α. Computation and I/O

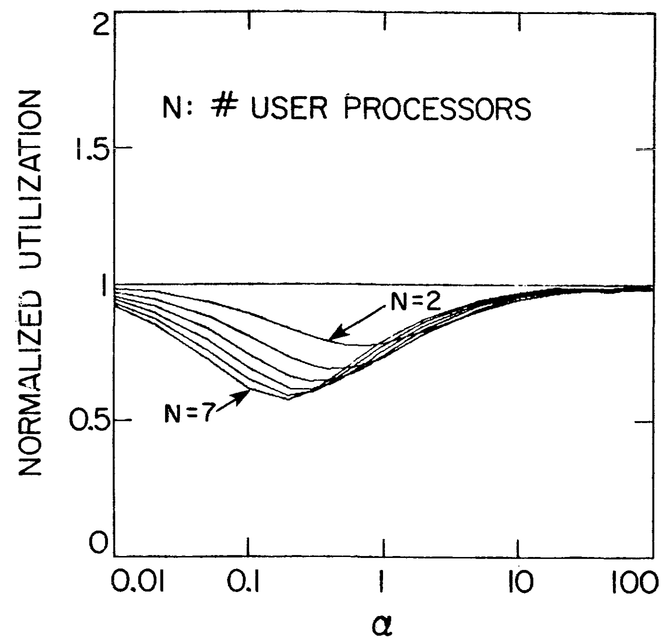


Figure 12—MP/C utilization, 2 to 7 processors

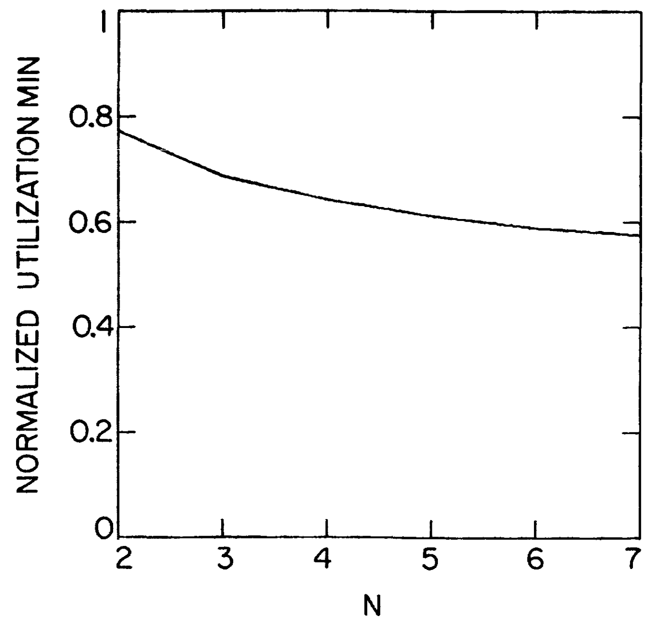


Figure 13—Lowest normalized MP/C utilization

times are about equal, and the extra delay of the MP/C is noticeable.

2. The dip deepens when N grows. Naturally, when there are more processors, more overhead is incurred.
3. The dip moves to the left as N grows. It takes smaller values of α to achieve the same degradation when there are more processors to contribute to the overhead.
4. The left side has an asymptote from above $\rho_{MP/C}/\rho_{star} = 1$. When I/O time is marginal, the effect of preemption is reduced, thus the MP/C and the star are getting closer to being equivalent. (They are equivalent for $\alpha = 0$.)
5. The convergence to the left asymptote is very slow. This is because another effect makes the MP/C slightly worse than the star. In the star, each job has to wait in queue only in the central node. In the MP/C model, a job has to repeatedly wait in queues in each of the intermediate stations. The additional queueing causes this small loss of utilization, which exists throughout the α range, but becomes the dominant factor only for small α.
6. For high values of α we observe the same asymptote, $\rho_{MP/C}/\rho_{star} = 1$. The kernel is the cause of congestion, and all jobs spend most of their time in the queue of the kernel. Hence, the difference between the MP/C and the star becomes unimportant. As observed above, both the MP/C and the star perform rather poorly for $\alpha > 1$. Thus, this part of the curves may be ignored altogether.

To summarize, note that at no point is the utilization of the MP/C better than that of the star. On the other hand, for all the cases tested, it was always more than half that of the star. This suggests that the MP/C, although it is much more limited in communication paths to the kernel than the star, has at least very similar performance. And recall that some of the other MP/C advantages over the star are that all processors require at most two links to other processors (whereas the kernel in the star requires N links), and that the MP/C can be decomposed recursively into smaller MP/C subsystems. We have suggested elsewhere under what circumstances the performance of the MP/C might be much better, as well as what the other MP/C advantages are.[1]

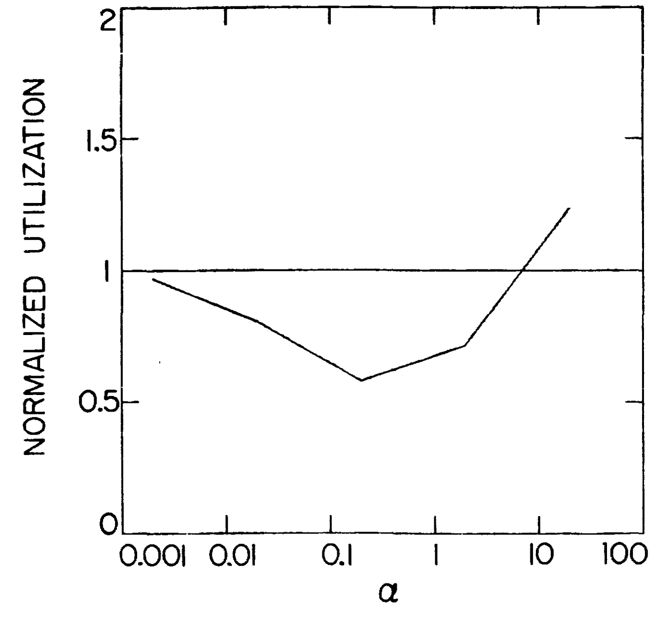


Figure 14—Normalized MP/C utilization, increasing α

Figure 13 shows one additional measure. The minimum relative utilization at the dip is plotted versus N, the number

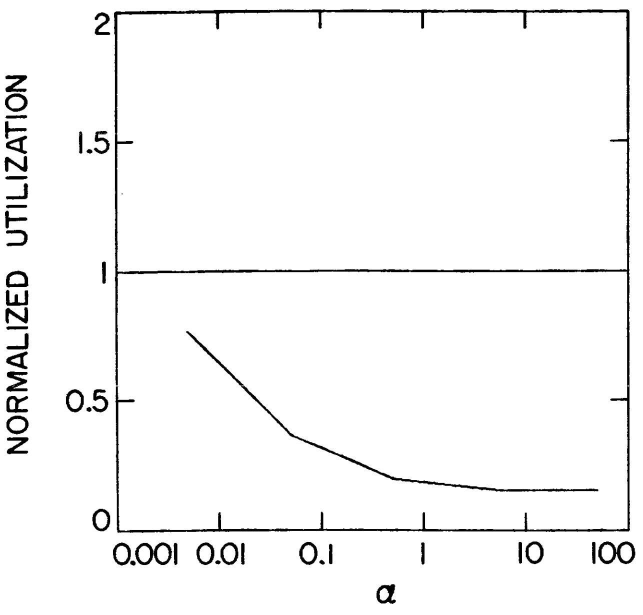


Figure 15—Normalized MP/C utilization, decreasing α

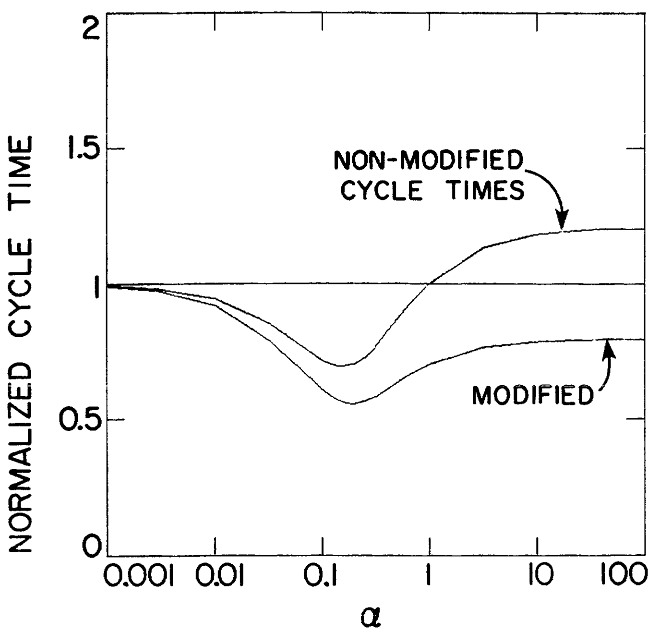


Figure 16—Normalized MP/C cycle times

of processors. This curve gives a lower bound on the performance of the MP/C, as compared to the star. Although it is speculative to conjecture how this curve might extrapolate, note that it shows a tendency to flatten out as N grows. This gives rise to the conjecture that even for higher N values the worst-case performance of the MP/C would not be much below that of the star. Also, note that star architectures with high N are rather infeasible in general, owing to the high degree of connectivity required of its central node, whereas no similar requirement applies to the MP/C, since it is a simple linear structure.

H.2. Processor Utilization—Nonuniform Case

As stated at the end of Section F, an interesting question is the performance when not all users pose the same workload requirements. Indeed, this is probably also the more realistic case for a general-purpose, multiple-user environment. We describe here the results of the modeling when the workload consists of a geometric series, where each processor has its α parameter twice that of its neighbor. We consider the cases both of an increasing series and of a decreasing one.

As suggested above, intuitively we would expect that the best arrangement is the decreasing one, that is, when processes with high α are allocated to processors near the kernel, and those with low α to the processors further away.

Figure 14 shows the utilization of the MP/C, relative to the star's, when the α values increase with i, the processor index. Figure 15 is the same for a decreasing series. The α values used in the graph are the average over all α_j, $0 < i \leqslant N$ (this is not really important, since same α is used for both the MP/C and the star, and a different calculation of α would only shift the graph sideways). The results for low α values resemble the uniform case above. However, for high values of α, we get results that are just the reverse of what might be expected. This phenomenon is commented on in the end of Section L, when it is compared with the simulation results (which do conform to intuition).

H.3. Cycle Times

As is explained in Section H, the first error introduced by the MP/C model causes overestimation of cycle times. Hence, it is reasonable to assume that the cycle times predicted by the model are only upper bounds on the real cycle times. The model prediction is depicted in Figure 16 by the curve marked "nonmodified cycle times."

In an attempt to derive the complementary lower bounds, we consider the following observation. The cycle time of the user job from P_i consists of the following:

1. Service time (mean = 1) and waiting time on P_i
2. Service time (mean = α) and waiting time on all P_j, $0 \leqslant j < i$, that is, on i processors

On the real MP/C, however, the cycle is composed of:

1′. Service time (mean = 1) and waiting time on P_i.
2′. Waiting time to seize the kernel.
3′. Service time on the kernel (mean = α).

We assume that item 1 models item 1, and that the sum of all waiting times in item 2 models item 2. Then the balance is only one service time on the kernel in the real MP/C, but i times the same service period in the model. This analysis suggests that the cycle times predicted by the model are too long.

Although this over-analysis is simplistic, we can use it to modify the cycle times by subtracting $i - 1$ kernel service periods from the total cycle time. This does not produce correct cycle times, but we assume that the modifying is an overshoot, and that it yields a lower bound.

Applying this procedure on the results described above, we obtain the modified cycle times curve in Figure 16. Thus, the model prediction provides an upper bound, and the modified cycle times are a lower bound. As we show in Section L.2, simulations (whose purpose is to validate these results) show that the modified cycle times are indeed an optimistic prediction.

I. ANOTHER ANALYTIC MP/C MODEL

We describe a second model for the MP/C. It is based on the method of *surrogate delays* of Jacobson and Lazowska,[16] as applied by Heidelberg and Trivedi[14] and on Cobham's analysis of priority queues.[9,18]

Each user processor, P_i, $1 \leq i \leq N$, has one job associated with it, J_i. This job cycles between P_i and the kernel. The kernel is a nonpreemptive priority queue, to model the fact that in the real MP/C the kernel serves the nearer processors first. Thus, J_i is assigned priority $N-i$. The cycle time consists of the following components:

1. Service time r_i at P_i, with mean of 1 unit
2. Delay d_i at P_i due to preemption by J_k, $i<k\leq N$
3. Waiting time s_i at the kernel
4. Service time α_i at the kernel, with mean α (either uniform for all J_i or variable)

Job J_i is preempted by J_k for α_k, the average time J_k is served by the kernel, in each cycle of J_k. J_i stays at P_i for $1+d_i$ units of time. During that time, J_k makes $\lambda_k(1+d_i)$ cycles, where λ_k is its throughput. This delay is accumulated for all $k>i$. Hence,

$$d_i = (1+d_i)\sum_{k=i+1}^{N} \alpha_k\lambda_k, \qquad 1\leq i\leq N. \tag{I.1}$$

Solving for d_i,

$$d_i = \sum_{k=i+1}^{N} \alpha_k\lambda_k \Big/ \Big(1-\sum_{k=i+1}^{N}\alpha_k\lambda_k\Big), \qquad 1\leq i\leq N. \tag{I.2}$$

The waiting time in the kernel, s_i, is composed of two identifiable parts, the mean residual life R of the job in service upon arrival of J_i at the kernel, and the time J_i has to wait for all jobs of higher priority to be served. The residual life has been shown to be:

$$R = \sum_{k=1}^{N} \lambda_k \overline{x_k^2}/2, \tag{I.3}$$

where $\overline{x_k^2}$ is the second moment of service time at the kernel for J_k. If we assume exponential distribution of service times with mean α_k, then the second moment is $2\alpha_k^2$, and

$$R = \sum_{k=1}^{N} \lambda_k\alpha_k^2. \tag{I.4}$$

During the waiting time s_i, any job J_k, $k<i$, with higher priority than J_i that arrives at the kernel is served before J_i. The delay incurred is α_k per cycle, and there are $s_i\lambda_k$ cycles of J_k during s_i. Hence,

$$s_i = R + \sum_{k=1}^{i-1} \alpha_k\lambda_k s_i. \tag{I.5}$$

Solving for s_i,

$$s_i = R\Big/\Big(1-\sum_{k=1}^{i-1}\alpha_k\lambda_k\Big). \tag{I.6}$$

We replace the queue at the kernel by a delay server (or infinite server) for each job with mean service (delay time for J_i of $s_i+\alpha_i$. We further replace the user processor P_i by a delay server with mean delay time of d_i+1. Thus,

$$\lambda_i = 1/(1+d_i+s_i+\alpha_i). \tag{I.7}$$

The model is solved by iteration. Consider the vector $\mathbf{x} = (d_1,\ldots,d_N,s_1,\ldots,S_N)$. The initial condition is

$$\mathbf{x} = 0. \tag{I.8}$$

The vector of throughput results of iteration m, $\vec{\lambda}^m = (\lambda_1,\ldots,\lambda_N)$, is used as input for the calculation of iteration $m+1$. Since $\vec{\lambda}^m$ is a function of $\mathbf{x}^m$ (as per eq. (I.7)), and since $\mathbf{x}^{m+1}$ is a function of $\vec{\lambda}^m$ (as per eq. (I.2), (I.6)), we can say

$$\mathbf{x}^{m+1} = \mathbf{f}(\mathbf{x}^m). \tag{I.9}$$

Thus, we solve a fixed point problem $\mathbf{x}=\mathbf{f}(\mathbf{x})$ in $2N$ dimensions. Note that equations (I.2), (I.6) are *not* guaranteed to produce stable, converging results always. In practice, we found that for $\alpha\leq 1$ equation (I.1) and (I.2) always converge. As will be shown in Section L, where the results are compared with simulation, this model produces fairly accurate results. However, for $\alpha>1$, the model converges to the wrong results. It is possible that this is due to the fact that the model is based on continuous-time analysis, rather than discrete. But recall again that the interesting range is $\alpha<1/N$.

J. SIMULATIONS: SIMULATION METHODOLOGY

Since the analytic modeling we have described involves some approximations, simulations are necessary to validate the results. We show that some, though not all, of the analytic results are validated by the simulations.

J.1. Simulation Language

In order to minimize the programming effort required, a simulation language is used. Of the languages that were available to us at the time, SIMSCRIPT[19] and GPSS,[15] the latter was chosen because of relative simplicity and ease of learning. The disadvantage of using a language is that its flexibility is limited; that is, it may not be possible to build an exact model.

J.2. Simulation Models

In the MP/C simulation model, a single job is associated with each user processor. That job holds the user processor for a period of time (referred to as *computation*) that is drawn from an exponential distribution. Then it moves to the kernel processor, and competes with other jobs for its service. Another fact should be considered: On the MP/C, if two or more jobs request service, the kernel serves the nearest processor first. To account for that in the simulation model, each user job is assigned a priority level higher than that of all processors further away from the kernel (that is, to its right). In other words, if P_0 is the kernel, the priority of P_i is $n+1-i$, $1 \leq i \leq n$.

When the kernel processor is seized by a user job from processor P_i, all the processors between P_1 and P_{i-1}, inclusive, are halted for the duration of service. The service time on the kernel is distributed exponentially, and its mean is α times the computation time on the user processors. α can be the same for all user jobs (the *uniform* case), or it can vary. Once service on the kernel is finished, all the halted processors resume operation, and the job returns to its user processor.

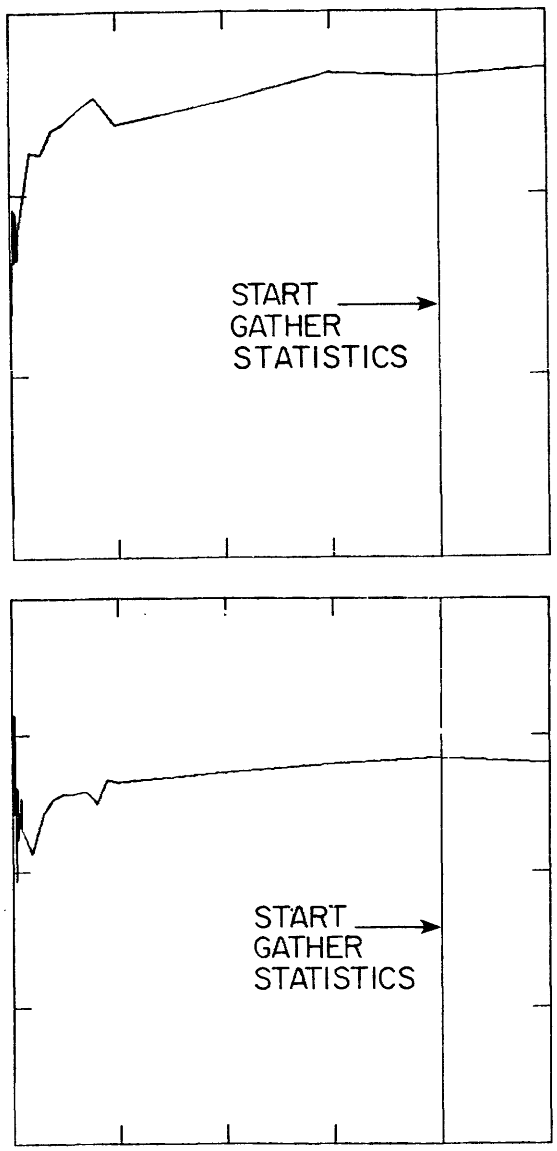


Figure 17—Simulation transient response

The star model is similar, with two differences. First, no halting of intermediate processors is necessary. Second, all user processors are assigned the same priority level.

We have listed both models elsewhere.[2]

J.3. Method of Statistical Sampling

In order to generate a large sample for each experiment, we use the method of *batches*. In this method the simulated system is first allowed to achieve its steady state, and then a single long run is divided into time periods, or batches, and statistics are gathered for each batch. This method needs less computing than the method of *independent replications,* in which the system is restarted for each replication, and thus the extra time for the decay of the transient response has to be repeated many times. The third known method, *regenerative simulations*, is not easily applicable here, as no relatively frequent regeneration Markovian state can be identified for either the MP/C or the star. These methods are discussed by Kobayashi and by Sauer and Chandy.[17,21]

J.4. Length of Transient Response

The length of the transient response was determined empirically from a series of pilot runs. Figure 17 represents the behavior of two of the performance measures used, cycle time (for P_1) and utilization (of P_7). These are accumulated statistics, such that every point in the graph represents the average of the observed measure from the beginning moment until the current time. The graph also shows the point chosen for the beginning of the first batch. It is clearly beyond most of the transient response.

J.5. Batch Length

The length of each batch was similarly determined by pilot runs. Figure 18 shows the behavior of the two measures during some long batches. In the beginning of each batch, all the statistics accumulated up to that point are purged, and accumulation is restarted. Hence, the sharp peaks at batch start represent random values, averaged over very short time periods, and the curves flatten out after some time. The batch length is chosen such that the transient effect of restarting the statistics accumulation has been damped. The batch end points are marked with vertical lines.

J.6. Number of Batches

The number of batches required is usually determined by the desirable confidence interval and by the need to make

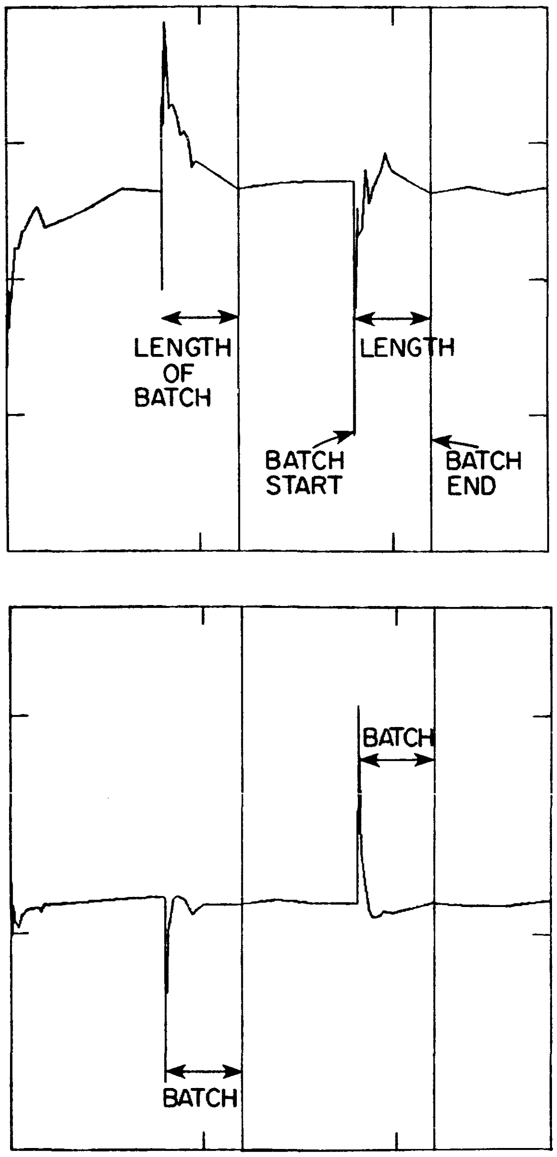


Figure 18—Simulation batches

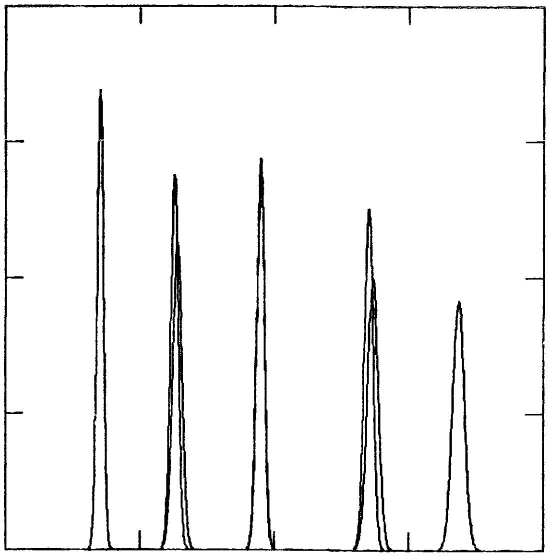


Figure 19—Utilization sample average distribution

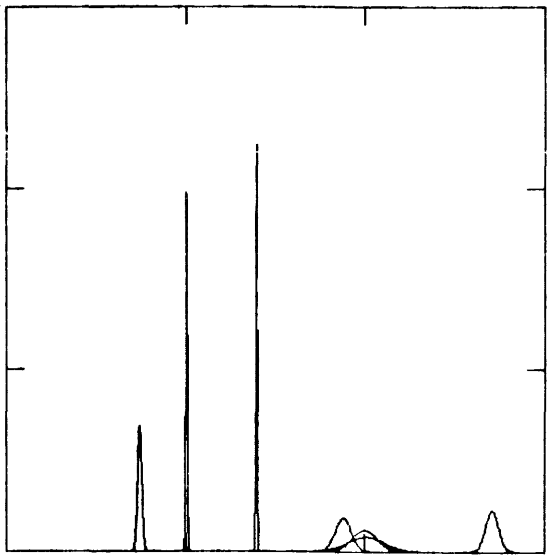


Figure 20—Cycle time sample average distribution

statistically significant distinctions between different quantities.[20] Also, to be able to claim that a sampling distribution is approximately normal (Gaussian), one usually needs at least 12 samples. Using a simulation run of 12 batches, we got results that were either almost identical, or separated well beyond any statistical doubt. Figure 19 shows the distribution density function of the sample averages of processor utilization of all the experiments made with either fixed $\alpha = 100$, or variable α with values around 100. Figure 20 is the same for cycle times. Note the narrow Gaussian "bells," the large distances between some, and the almost complete overlap of others. Note also that these are not the density functions of the distributions of the measures examined (they need not even be normal), but rather the distributions of their sample averages.

Table I—Comparison of pilot simulation and analytic results, star and MP/C with 7 user processors

α	Utilization, $\frac{MP/C}{Star}$		Cycle Time, $\frac{Star}{MP/C}$	
	Pilot Simulations	Analytic Model	Pilot Simulations	Analytic Model
1000	1.0	1.0	0.76	1.2
100	0.9	1.0	0.76	1.2
1	0.9	0.79	0.66	1.0
0.4	0.89	0.65	0.73	0.81
0.01	0.97	0.92	0.97	0.99
0	1.0	1.0	1.15	1.0

K. PILOT SIMULATION RUNS

The first task of the simulation was to verify the general behavior of the analytic model, across different workloads (α values), for uniform α (i.e., same α for all user jobs). Pilot runs (i.e., sample size of one) were made at various α values. The results are summarized in Table I, and compared to the result of the first analytic model (Sections E through G). Although they do not constitute a statistical proof, the simulations are close enough to the modeling results to suggest that the analytic model provides reasonable results.

L. DETAILED SIMULATION

L.1. Results

The results of some detailed simulation runs are shown in Figures 19 and 20. As explained above, it is clear from those figures that ignoring confidence intervals and considering only the averages would not cause grave mistakes. Table II summarizes the simulation results for α = 0.1, 1, 100. It also compares them to the results of the analytic model described in Sections E through G (both modified and nonmodified cycle

Table II—Comparison of simulation and analytic results, star and MP/C with 7 user processors

	Cycle Time			Utilization		
Load	Star	MP/C	Star / MP/C	Star	MP/C	MP/C / Star
Simulations, α=0.1						
uniform	1.17	1.47	0.796	0.8484	0.692	0.816
Analytic Models (absolute and relative to simulations), α=0.1						
uniform, modified	1.18 (101%)	1.65 (112%)	0.716	0.8442 (99%)	0.52 (75%)	0.616
uniform, non-modified	"	1.95 (133%)	0.606	"	"	"
uniform, model 2	"	1.50 (91%)	0.787	"	0.6674 (96%)	0.791
Simulations, α=1						
uniform	6.95	9.60	0.724	0.14376	0.13521	0.940
Analytic Models (absolute and relative to simulations), α=1						
uniform, modified	7.00 (100%)	7.00 (73%)	1.00	0.1428 (99%)	0.1129 (84%)	0.791
uniform, non-modified	"	10.00 (104%)	0.700	"	"	"
uniform, model 2	"	10.95 (114%)	0.639	"	0.09136 (68%)	0.640
Simulations, α=100 and variable α						
uniform	698.1	940.4	0.742	0.001366	0.001351	0.989
increasing	502.7	998.6	0.503	0.00095	0.000639	0.672
decreasing A	502.7	371.6	1.352	0.00095	0.001686	1.774
decreasing B	1355.2	1001.2	1.353	0.000352	0.000628	1.783
Analytic Models (absolute and relative to simulations), α=100 and variable α						
uniform, modified	699.8 (100%)	581.9 (62%)	1.202	0.001429 (105%)	0.001424 (105%)	0.996
uniform, non-modified	"	882.0 (94%)	0.793	"	"	"
increasing	28.8	495.2	0.058	0.009964	0.014860	1.491
decreasing	1772.4	305.8	5.795	0.009964	0.006151	0.617

times) and (for $\alpha \leq 1$) the second model, Section H. Note that when comparing utilization, we consider the MP/C utilization divided (normalized) by that of the star; however, when considering cycle time, we use the MP/C cycle time in the denominator instead. This is because the inverse of the cycle time (rather than cycle time iteself) is the measure of goodness here. Thus, for both measures, the higher the ratio, the better the MP/C as compared to the star. The entries marked "uniform" are the results with uniform workload on all user processors. The increasing case is when $\alpha_1 = 4$, $\alpha_2 = 8, \ldots,$ $\alpha_7 = 256$ (α_1 is the load on P_1, the processor closest to the kernel, etc.). This powers-of-two series of α values was chosen such that the load on the kernel (in terms of average time per job on the kernel) is similar to the "uniform," $\alpha = 100$ case. As expected, the MP/C in this case is worse (relative to the star) than in the uniform case. The decreasing case has two variants. In variant A, the loads are the reverse permutation of the previous case. That is, $\alpha_1 = 256$, $\alpha_2 = 128, \ldots, \alpha_7 = 4$. This time the MP/C performance is better than the uniform, $\alpha = 100$ case. However, the MP/C performance is so improved that the load on the kernel is significantly reduced. Hence, variant B was constructed with the following geometric series of α values: 6912, 3456, 1728, 864, 432, 216, 108, such that the average time per job on the kernel resembles that of the uniform, $\alpha = 100$ and the increasing cases. Nevertheless, the ratios are the same as in variant A.

The MP/C utilization can also be compared to the lower bound provided by the first MP/C deterministic model of Section D:

$$\rho_{i,\,\mathrm{MP/C}} = \frac{\beta}{\beta + N} = \frac{1/\alpha}{1/\alpha + N}.$$

For $N = 7$ and $\alpha = 0.1$, 1, 100, $\rho = 0.59$, 0.125, 0.00143, respectively. The simulation showed slightly better performance (than this theoretical lower bound) for $\alpha = 0.1$, 1 but worse performance for $\alpha = 100$.

L.2. *Discussion*

One question naturally arises when these data are examined. Since the MP/C inherently demonstrates a priority mechanism, will a star with priorities behave like the MP/C? If not, what makes the difference? Introducing priorities to the star model takes effect only for high α. There, it does not make the star imitate the MP/C. Rather, it causes livelock, or partial starvation. That is, the two or three jobs with the highest priorities get to use the kernel almost exclusively, whereas the lower priority jobs starve forever. The reason for this is best demonstrated in specific terms. Recall that for $\alpha = 100$ each cycle consists of, among other things, one unit of time (on the average) in the user processor and 100 units of service time in the kernel. Suppose the job of user 1 has just released the kernel, and the job of user 2 (having the next highest priority) seizes it. In the star, when the job from P_2 uses the kernel, P_1 is free to complete its computation part of the cycle and is ready to grab the kernel again once user 2 releases the kernel. Thus, most of the time the kernel is shuffled back and forth between P_1 and P_2, and the other users starve waiting for the kernel service. On the MP/C, when P_2 uses the kernel, P_1 is preempted. Thus, there is higher probability that a processor other than P_1 will be ready to grab the kernel once P_2 releases it. In short, P_1 can execute the (relatively small) computation part of its cycle only when no other processor uses the kernel. For seven processors and $\alpha = 100$, this is very rare indeed. On the other hand, P_7 of the MP/C, although it has the lowest priority, is never interrupted by anyone. This effect causes the *inversion anomaly*. In the simulations, the better-served processors are those further away from the kernel, the opposite of what may intuitively be expected. This anomaly of course can not be detected by the analytic model of Section F.

The inversion anomaly can be countered by a more elaborate preemption mechanism, for example, a minimal delay between successive preemptions, and by optimally arranged nonuniform workload.

The detailed simulation enables us to detect the inaccuracies of the analytic models. The two models show similar fit for $\alpha \leq 1$. Nonmodified cycle times of the first model are more accurate than modified cycle times for $\alpha \approx 1$ but less so as α decreases. As was explained in Section H, the second model does not work for $\alpha > 1$.

The reversal of the utilization figures at high α is an amplified version of the inversion anomaly. It seems to stem from the replacement of simultaneous preemption by successive service periods. Consider for example the increasing case. In the real MP/C, as well as in the simulations, when the user job from P_7 holds the kernel, no other processor is active, and recall that $\alpha_7 = 256$, the highest. In the analytic model, on the other hand, all processors, except P_7, are active at that time. Thus, the utilization in the increasing analytic model is higher than in the simulations. The error in the decreasing case is harder to figure out. The above explanation would have caused the same, rather than the opposite, effect.

Note, again, that these discrepancies are noticeable only at high α. Consider Figures 11 and 16 and Table I. For $\alpha < 10$ it seems that both simulations and analytic modeling yield similar results, in both uniform and nonuniform cases. Since the analytic model breaks down only at high α, and since, as it was emphasized above, this range of workload is not really important, we can ignore this problem.

M. CONCLUSION

We have discussed the MP/C performance under various circumstances. First, the performance of tree-structured algorithms was discussed. That application seems to be the most suited to the MP/C, and the MP/C performs probably better than any other architecture.

Next we examined the other side of the spectrum, the application of the MP/C to multiprogramming general purpose computations. This case serves as a worst/case, or lower/bound, analysis, for the MP/C. We have devised some analytic modeling techniques. Due to the complexity of the MP/C, only approximate analytic modeling was possible. Simulations were used to verify the results of the analytic modeling and to estimate their error.

We have found that the analytic models provide acceptable rough estimation of performance. Also, it was shown that, in the worst case, the performance of the MP/C is similar to that of the star multicomputer. That is, the relative advantages of the MP/C do not necessarily come at the expense of performance.

As we have suggested elsewhere,[1] under the right conditions the MP/C has the potential to perform much better than other architectures. We are currently performing additional evaluation, comparing the MP/C to multicomputers using local networks and to multiprocessors using shared memory, by means of detailed emulations, simulations, and a complete implementation.

ACKNOWLEDGMENTS

We are grateful to Steve Bruell and Gianfranco Babo, who let us use PNET, and to Neil Polhemus, for helping with the simulations.

REFERENCES

1. Arden, B.W., and R. Ginosar. "MP/C: A Multiprocessor/Computer Architecture." *IEEE Transactions on Computers,* C-31 (1982), pp. 455–473.
2. Arden, B.W., and R. Ginosar. "Performance Evaluation of the MP/C," Technical Report 294, Dept. of Electrical Engineering and Computer Science, Princeton University, Princeton, New Jersey.
3. Arden, B.W., and H. Lee. "Modeling Regular, Process-Structure Networks." *AFIPS, Proceedings of the National Computer Conference* 1979, Vol. 49, pp. 95–102.
4. Anastas, M.S., and R.F. Vaughan. "Limiting Multiprocessor Performance Analysis." *Proceedings of the 1979 International Conference on Parallel Processing,* pp. 58–64.
5. Bruell, S.C., and G. Balbo. *Computational Algorithms for Closed Queueing Networks.* Amsterdam: North-Holland, 1980.
6. Balbo, G., S.C. Bruell, and H.D. Schwetman. "Customer Classes and Closed Network Models—A Solution Technique." *Proceedings of the IFIP Congress* 1977, Amsterdam: North Holland, pp. 559–564.
7. Baskett, F., K.M. Chandy, R.R. Muntz, and F.G. Palacios. "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers." *Journal of the ACM,* 22 (1975), pp. 248–260.
8. Clark, W. *The Gantt Chart* (3rd ed.). London: Sir Isaac Pitman & Sons, 1952.
9. Cobham, A. "Priority Assignment in Waiting Line Problems." *Operations Research,* 2 (1954), pp. 70–76.
10. Chang, D.Y., D.J. Kuck, and D.H. Lawie. "On the Effective Bandwith of Parallel Memories." *IEEE Transactions on Computers,* C-26 (1977), pp. 480–489.
11. Enslow, P.H. Jr. "Multiprocessor Organization—A Survey." *ACM Computing Surveys,* 9 (1977), pp. 45–71.
12. Greenberg, S. *GPSS Primer.* New York: John Wiley, 1972.
13. Heidelberg, P., and K.S. Trivedi. "Analytic Queueing Models for Programs with Internal Concurrency," Technical Report RC 9194, IBM Watson Research Center, January 1982.
14. *General Purpose Simulation System V User's Manual.* IBM SH20-0851, IBM, 1970.
15. Jacobson, P.A., and E.D. Lazowska. "Analyzing Queueing Networks with Simultaneous Resource Possession." *Communications of the ACM,* 25 (1982), pp. 142–151.
16. Kobayashi, H. *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology.* Boston: Addison-Wesley, 1978.
17. Kleinrock, L. *Queueing Systems, Vol. 2.* New York: John Wiley, 1976.
18. Kiviat, P.J., R. Villanueva, and H.M. Markowitz. *The Simscript II Programming Language.* New York: Prentice-Hall, 1968.
19. Miller, I., and J.E. Freund. *Probability and Statistics for Engineers.* New York: Prentice-Hall, 1965.
20. Satayanarayanan, M. "A Survey of Multiprocessing Systems." Technical Report RC7346, IBM Watson Research Center, October 1978.
21. Sauer, C.H., and K.M. Chandy. *Computer System Performance Modeling.* New York: Prentice-Hall, 1981.
22. Sauer, C.H., and E.A. MacNair. "Queuing Network Software for System Modeling." *Software—Practice and Experience,* 9 (1979), pp. 369–380.

A multiprocessor with replicated shared memory

by SIGURD L. LILLEVIK and JOHN L. EASTERDAY
Oregon State University
Corvallis, Oregon

ABSTRACT

A multiprocessor includes five 8086 microprocessors interconnected with replicated shared memory. Such a memory structure consists of a set of memories, one for each processor, with identical contents. This minimizes read interference since each processor simply accesses its own private copy of the shared memory. To ensure shared memory integrity, write requests transfer data over the MULTIBUS to all copies in parallel. Overall, replicated shared memory structures provide improved concurrency.

An HP 64000 Logic Development System serves as a host computer for program development and a bulk storage device. A power-on and restart monitor in shared PROM provides a run-time debug and method for down-loading the operating system and application programs. The real-time, multi-tasked operating system (called MPX) distributes a sequence of high and low priority tasks, with possible preemption, among the processors. MPX floats from processor to processor while balancing the system load for maximum concurrency and throughput.

INTRODUCTION

In the design of multiprocessor computers, a shared memory often links together the individual processors. This facilitates coordination of the system and assures a convenient communication medium for the processors. But shared memory and all other shared resources contributed to degradation of the system through interprocessor interference; simply stated, not all requests for a shared resource may be honored at one instant in time. Recently, the principle of using several copies of the shared memory, one per processor, instead of just a single memory has been investigated. Such a replicated shared memory (RSM) structure offers the advantage of minimal read conflicts since individual processors access their own private copy of the shared memory. So that each copy remains the same, writes to the RSM must modify all copies and this requires some synchronization and arbitration of requests. Overall, an RSM structure provides improved concurrency for an increase in system throughput. Applications such as artificial intelligence, Monte Carlo simulation, solution of partial differential equations, and others that require extremely high computing rates may ultimately benefit from RSM structures.

This manuscript describes a multiprocessor used to study replicated shared memory techniques. Specifically, researchers wish to determine the applications and problem characteristics best suited for multiprocessors with RSM and the expected increase in system throughput. In the next section of this paper, a review of previous work defines the origin and present status of replicated shared memory. The hardware design of an RSM multiprocessor will be described in terms of major component selection and their interaction. Then, the next section outlines the overall software design and describes the programming environment and operating system. The last section summarizes the paper, provides design guidelines for RSM structures, and concludes with some suggestions for further research.

BACKGROUND

The development of a computer class has traditionally followed a wheel of reincarnation as suggested by Myer and Sutherland.[14] With multiprocessors, technology (today VLSI) drives the wheel or evolutionary chain. Thus, designers have more recently looked to multiprocessor structures because of their potential for both speedup and generality. Satyanarayanan[16] has prepared an annotated bibliography that surveys the field of existing multiprocessor machines. He suggests several possible dimensions to the multiprocessor design space: Are the processors symmetrical or asymmetrical? Do the processors contain local resources? What interconnection network is required? Are the input and output (I/O) devices shared or local? How do the processors communicate, interrupts, mailboxes, etc.? These and other questions must be answered in the design of a multiprocessor computer

Experience with multiprocessor computers has shown designers that the key to efficient exploitation of parallelism is interprocessor communication. And this relies on a cost-effective interconnection network. In a survey article by Feng,[6] the interconnection design space is dictated by the mode of operation (synchronous or asynchronous), control strategy (centralized or distributed), switching methodology (circuit or packet), and the topology (static or dynamic). Another classification scheme by Anderson and Jensen[1] delineates various levels of interprocessor message handling and hardware topologies to define structures. In a paper on highly parallel computing, Haynes, et al.[8] summarize several practical interconnection networks using three parameters: cost, generality, and efficiency. Very efficient networks include crosspoint switches and systolic arrays (see Kung).[10] The crosspoint represents a very general but expensive switch while the systolic array is specialized and inexpensive because it is very regular and well-matched for VLSI production. Some less efficient and less expensive interconnections ranging from the general purpose to specialized include the banyan,[7] k-cube,[12] shuffle,[18] and tree.[5] Finally, the ring topology provides an inefficient but low-cost and general interconnection network. Some trends may be easily deduced concerning interconnections, the greater the efficiency and generality, the greater the cost. Overall, the entire interconnection network space includes many possibilities and most designs consider such additional constraints as basic problem dependencies, fault tolerance, and implementation realities.

If a multiprocessor typically contains a shared memory for interprocessor communication, then a multiport memory may be used and elaborate interconnection networks avoided. The difficulty of multiport memories is in the implementation. To provide several ports to a memory, a design requires several sets of address, control, and data buses. Other implementation difficulties include simultaneous read and writes, or multiple writes to the same memory location. In addition, the question of synchronous or asynchronous access always occurs in memory design and requires investigation. Overall, the implementation of multiport memory presents many problems.

Several researchers have studied methods for the design of multiport memory. For example, Chu and Korff[3] have reported on a multiaccess memory that consists of an array of cross-coupled gates with separate sense and drive lines for each port. They resolve the problem of multiple writes to the same memory location with a set of comparators that determine the port to receive priority. All other ports must wait and this introduces additional memory access delay. A similar design by Chang[2] provides a multiport memory for reads and a conventional single-port memory for writes. Thus, write requests must be arbitrated externally. Chang contends that many applications require only a single write port and that the multiport read characteristics will be sufficient. Chu and Korff's and Chang's memories are intended for implementation as integrated circuits.

Rather than provide multiple ports to a memory, concurrency may also be developed using several copies or replication of the memory contents. Both Covo[4] and Pearce and Majithia[15] have suggested that memory replication, a copy for each port, may be used as a multiport memory structure. More recently, Lillevik, *et al.*[11] have presented guidelines for the design of multiport memory using replication techniques. In this design, reads from memory may occur asynchronously, concurrently, and at various cycle times. By eliminating read conflicts, interprocessor interference is reduced and throughput increased.

A multiprocessor with replicated shared memory (RSM) is shown in Figure 1. It consists of several processors each with a copy of the common memory, two sets of bus switches, and a single multiport bus. This multiprocessor and memory function as follows: when a processor wishes to read from the RSM, Switch A is enabled and Switch B is disabled, and data in the common memory are read by the processors. Writes proceed differently depending on the port wishing to write to the RSM. Here, the write port enables both Switch A and B, while the remaining, passive ports disable Switch A and enable Switch B. Data from the writing processor pass through Switch A and all Switch B's so that every copy of the common memory remains the same. Because several processors may wish to write simultaneously, write requests require arbitration. Skinner and Lillevik[17] have described an implementation of this design intended for aerospace, remote, and process control applications. It uses three CMOS microprocessors, transmission gates for bus switches, and a synchronous multiport bus with distributed arbiters for control. They point out that each processor, memory, and switch section could reside on a single chip with the multiport bus signals terminating on pins. Such a chip represents a versatile building block for multiprocessor computers.

HARDWARE DESIGN

At Oregon State University (OSU), a five processor computer has been developed and is in operation to investigate the characteristics of replicated shared memory. Selection of major components was based on technical, economic, and political criteria. For example, the growing industry-university cooperation at times dictates the use of specific products. Perhaps the decision to use the MULTIBUS as a system bus impacted the overall design more than any other tradeoff. This bus supports multiple masters (which are active) and multiple slaves (which are passive). Basically a common bus with synchronous and distributed arbitration, the MULTIBUS represents a widely accepted bus under consideration as a standard. In the replicated shared memory design, the individual memory copies act as resident devices for reads, and slaves for writes. Once the system bus was selected, use of the 8086 microprocessor assured compatibility between the bus and the processors. As a third generation microprocessor, the 8086 consists of a 16-bit general purpose machine with a 1 Mbyte, segmented address space. Also, the 8086 contains some multiprocessor features such as bus lock, synchronization, and coprocessing.

A typical MULTIBUS/8086 master implementation may range in complexity, but the general approach used in this design is illustrated in Figure 2. Interface circuitry splits the 8086's buses into a resident and system configuration by decoding the address space. These circuits include an inter-

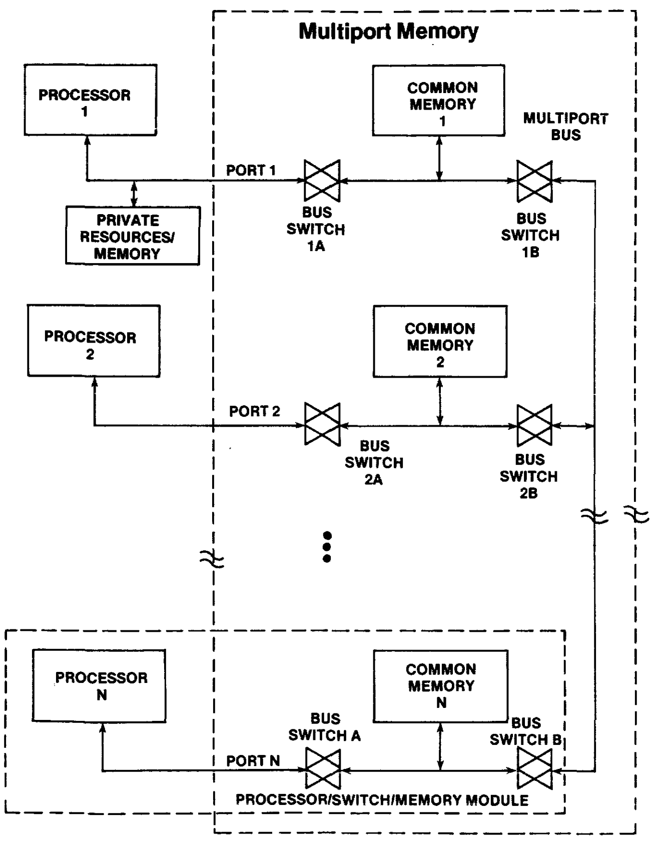


Figure 1—Replicated shared memory structure

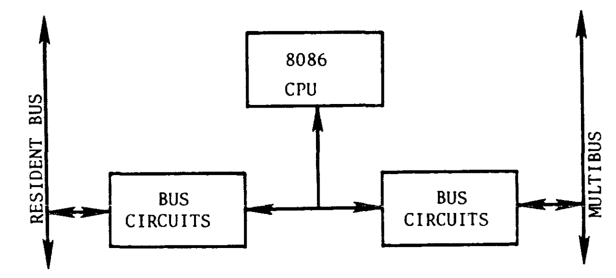


Figure 2—Typical block diagram of MULTIBUS/8086 master

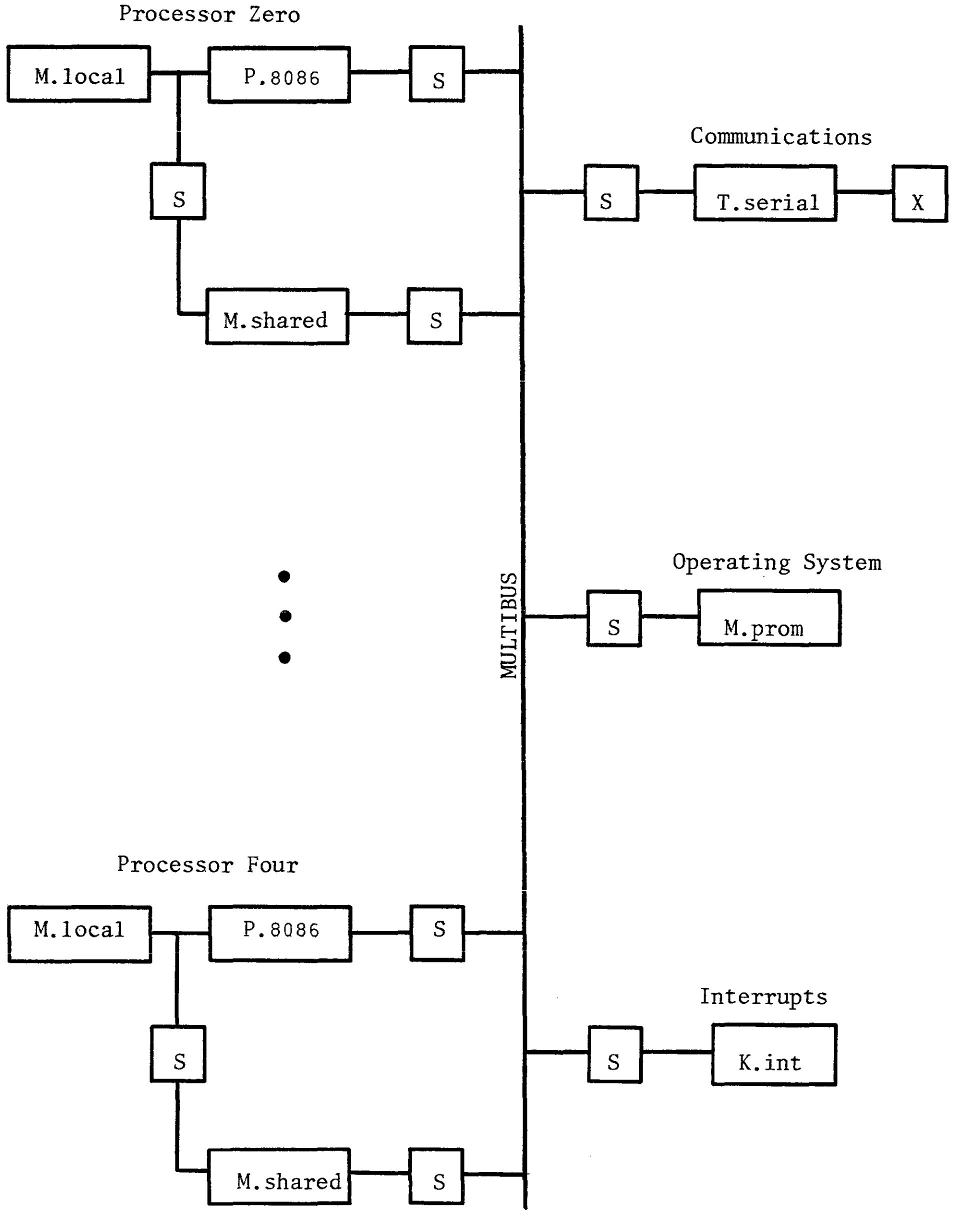


Figure 3—PMS diagram of a multiprocessor with replicated shared memory

rupt controller, arbiter, command controller, address latch, and data transceivers. Slave devices do not contain these circuits, but instead they simply monitor the MULTIBUS address and control buses and respond to commands as they occur.

A PMS diagram of the OSU multiprocessor computer is shown in Figure 3. The system contains five processor boards and one communications board. Each of the five processor boards includes an 8086 microprocessor, up to 256 Kbytes of dynamic RAM, and MULTIBUS interface support circuits. The RAM may be partitioned into all local, all shared, or some division of local and shared memory. Besides providing five RS232 ports to the external world, the communications board contains 16 Kbytes of programmable, read only memory (PROM) which hold a power-on and restart monitor program shared among the five processors. Also, the communications board implements an interprocessor interrupt scheme over the MULTIBUS that allows any processor to interrupt another processor (including itself). Finally, this board contains the MULTIBUS priority resolution circuitry consisting of a priority encoder/decoder pair.

Some deviations from a typical MULTIBUS/8086 implementation were required for this design. First, not only did the address space require decoding to form a resident and system bus, but the read/write space had to be included in the decoding. To obtain this information, the design used the processor status lines that indicate read and write operations. Another problem involved the dynamic RAM controllers. They assume a single-port memory system, while a dual-port memory system was required. An additional finite state machine arbiter was used to resolve requests for the memory and generate transceiver enables for the two address and data buses. A fundamental and essential multiprocessing feature was partially disabled by decoding the read/write space; the facility for a read and modify cycle and semaphores. Since reads to shared memory occur on a resident bus, the system bus lock signal was disabled. The solution involved distribution of individual processor generated locks to all dynamic RAM arbiters. These three problems represented only minor changes from a typical MULTIBUS/8086 application and generated few additional chips (SSI, MSI) on the boards.

Physically, the multiprocessor consists of a single, desk-size 19" rack holding a power supply, 21-slot MULTIBUS card cage and back plane, and a panel with RS232 connectors and processor/system restart switches. The five processor boards each contain around 80 packages and 2,000 wirewraps. Finally, the last and sixth board contains the serial ports, PROM, and interrupt generator circuits.

SOFTWARE DESIGN

The software environment of the multiprocessor includes a host computer system with the multiprocessor as an attached processor. In this configuration, the host computer, an HP 64000 logic development system, functions as a program development system and a bulk storage device. Here, the host connects over a low-speed, RS232 serial port shared among the processors.

For hardware, the host supports a 12 Mbyte hard disk, a line printer, three CRT terminals each with dual floppy disks, a PROM programmer, a logic analyzer, and an in-circuit emulator. To assist in program development, the host software includes a file manager, editor, assembler, PL/M and Pascal compilers, and a linking loader. With this support on the host system, users prepare object files for later down-loading to the multiprocessor and eventual execution.

Upon power-on or a system restart, the processors execute a monitor program resident in shared PROM. Besides down- and up-loading of object files from and to the host, the monitor contains a run-time debug program. The command syntax parallels that developed for Intel's SBC 86/12[9] but with an additional processor field appended to each command. For example, one may request register or memory information about a single processor or a group of processors. The same holds true for breakpoints and the single-step mode. With this monitor, users may down-load and execute the operating system.

Since programmers develop software external to the machine, the multiprocessor requires only a minimal operating system. In addition, the purpose of this research focuses on the principle of replicated shared memory structures, not multiprocessor operating systems. Thus, the operating system functions primarily as a multiprocessor task multiplexer and distributes a sequence of tasks among the individual processors. Denoted MPX for multiprocessor executive, the operating system consists of a multiprocessor version of an Intel application note[13] on multitasking for a single 8086. MPX allows for real-time task distribution of both high- and low-priority tasks with possible preemption. Tasks enter first-in, first-out (FIFO) queues and may be delayed for several milliseconds before assignment to a processor.

The operating system resides in shared memory with one processor running executive code and the remaining processors running associative code. Once a processor begins execution of a task, it enters a run state. Thus, the state diagram of a multiprocessor contains three states: executive, associative, and run as shown in Figure 4. At any instant in time, only one processor may enter the executive state with the others in the associative state or run state. When the executive assigns itself a task, it releases ownership of MPX and if another processor becomes idle, the new processor takes ownership of MPX as the new executive. Then, later the old processor becomes an associative. In summary, MPX allows the executive to float from processor to processor.

As the executive, a processor continuously monitors the high- and low-priority task queues and distributes the tasks. In addition, the executive preempts low-priority running tasks, maintains the queues as it assigns tasks, updates the task directory, and if it assigns itself a task it releases MPX ownership and executes the task. Concurrently, associative processors continuously check if the executive assigned them a task, then either execute the task first, or then check if the executive has been released. If released or unlocked, a processor asserts ownership of the executive and continues distributing tasks. The executive may also change location as processors leave the run state. Otherwise, processors return to their original state: executive or associative.

Once a task becomes listed in the task directory, it may follow a sequence of state transitions as shown in Figure 5. First, an idle task corresponds to one not queued, running, or preempted. In all cases, a task must pass through either the high- or low-priority queue before assignment by the executive for execution on a specific processor. From the idle state, a task may enter the queued state for latter assignment, or it may enter the wait state for a specified number of milliseconds and then enter the queued state. When assigned to a processor

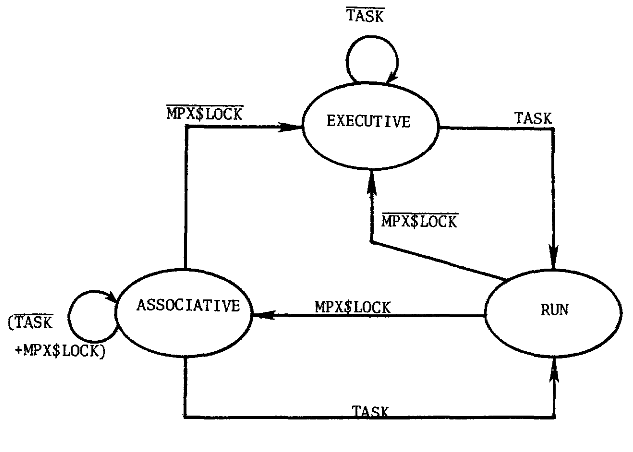


Figure 4—Individual processor state diagram

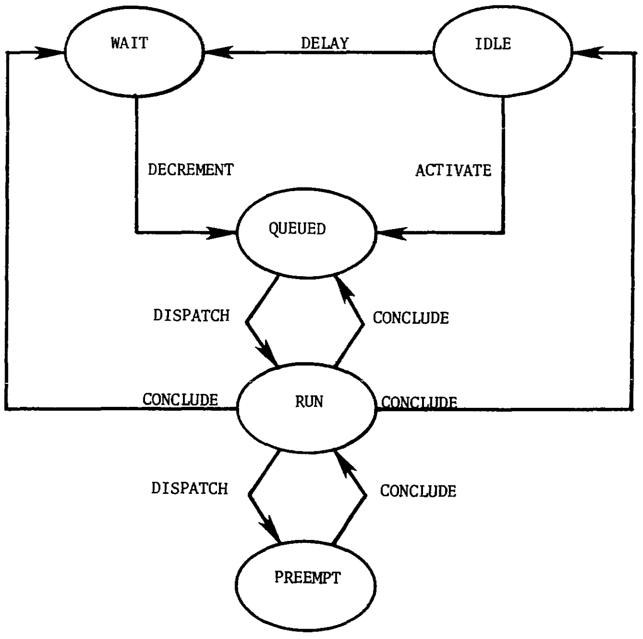


Figure 5—State diagram of an MPX task

for execution, a task enters the run state unless preempted by a high-priority task. At this time, the low-priority task enters the preempt state until the executive may no longer assign high-priority tasks. When this occurs, the preempted task continues execution. Interrupts facilitate task preemption. Following the run state, a task may reenter the wait or queued states, or simply the idle state.

Several procedures, accessible by all processors and located in shared memory, manipulate the execution of tasks. As with many shared structures, semaphores protect the data or task directory, but not the code or procedures which alter the data. To post a task in the task directory, the define/remove procedure is called. Each entry in the directory consists of a name, start address, local or shared task, high- or low-priority, and status information. Then, either the activate or delay procedure enters an idle task into the appropriate queue with possibly some delay if delay was used. Primarily, the executive repetitively calls the dispatch procedure that actually assigns queued tasks to specific processors. First, dispatch satisfies all entries in the high-priority queue before considering the low-priority queue. Then, the task at the head of the queue is checked to see if it is a shared or local task. If it is shared, then dispatch determines if a processor is available or can be preempted and assigned the task. If it is a resident task, then the requested processor is checked to see if it is available or may be preempted and assigned the task. When a task at the head of a queue may not be assigned to a processor, then dispatch returns it to the appropriate queue. Once dispatch satisfies both queues, then it returns and the executive decides if it has assigned itself a task. Upon completion of a task, the conclude procedure is called that terminates the task and either returns it to the idle state, places it in a queue, or the waiting list. An interrupt procedure, decrement, occurs every millisecond to update the waiting list of tasks. When a waiting task times out, decrement enters the task into a queue. Together, these elementary procedures coordinate the computing activities of the multiprocessor.

Certain constraints limit the flexibility and potential concurrency of the multiprocessor. First, all tasks must reside in memory, either shared or local, prior to being listed in the task directory. Because the hardware does not contain any fast bulk storage, the multiprocessor does not implement a memory management system. This explains the reason for resident tasks. Second, to obtain the maximum concurrency of the multiprocessor, one must exercise judicious care in both the distributions of tasks and their sequence of activation. Task distribution involves their location in memory, shared or local, and the priority assigned to them. As with other multiprocessors, the sequence of task activation depends largely on the application. Yet under proper conditions, MPX achieves much parallelism with the multiprocessor.

CONCLUSION

This manuscript has described the hardware and software design of a multiprocessor with replicated shared memory. Such a shared memory structure consists of several copies of the memory, one for each processor, so that individual processors read from their own private copy. To maintain the same information in all copies of the memory, all writes must modify all copies. Thus, reads may occur concurrently, asynchronously, and to different addresses. Overall, multiprocessors with replicated shared memory experience reduced interprocessor interference for an increase in throughput.

The multiprocessor discussed in this paper contains five 8086 microprocessors, each with up to 256 Kbytes of dynamic RAM for a maximum system memory of over 1.2 Mbytes. Use of the MULTIBUS facilitates interconnection of the processors to each other and to shared resources. In addition, the MULTIBUS provides a medium for writes to the replicated shared memory copies. When a write occurs, all the memory copies become slave devices and accept the data on the bus. The shared resources include five RS232 serial ports, PROM holding a power-on and restart monitor program, and an interrupt generator for interprocessor interrupts. One of the serial ports connects to a host computer, an HP 64000 logic development system, used for program development and as a bulk storage device. Also, the host includes a 12 Mbyte hard disk and three CRT terminals each with dual floppy disks. The multiprocessor monitor program provides for down- and uploading of object files to and from the multiprocessor.

With the multiprocessor monitor one may down-load into shared memory and execute the operating system MPX (multiprocessor executive). This real-time multi-tasked system allows for both local and shared tasks with either high or low priority. MPX may float from processor to processor and primarily functions to empty the two queues of tasks on a FIFO scheme. Tasks are assigned to individual processors and low-priority tasks are preempted, if possible, by high-priority tasks. With this system judicious care must be exercised in the distribution of tasks in memory and their order of activation.

For many cases, MPX balances the processor loads and the system achieves good concurrency and parallelism.

When designing a system with replicated shared memory one must consider several parameters, each with many tradeoffs. Perhaps selection of a system bus represents the one decision with the greatest impact. And once the bus has been selected the processor should be determined by the extent of support for the bus. Not crossing family boundaries helps to assure a good match of bus and processor. Certainly, the processor's facilities for multiprocessing enter into the selection process as well. Finally, the design of the operating system requires considerable thought. To fully exploit the parallelism inherent in the hardware, the operating system must be flexible and able to adjust to widely varying loads and degrees of concurrency.

Like all computer designs, one determines at a later date what additional features and characteristics might improve the original system. As for the multiprocessor described in this paper a redesign might include a further split of the 8086 bus into several MULTIBUSes and not just a system bus and a resident bus. This would facilitate creation of a hierarchy of processors interconnected with several replicated shared memory structures. Many geometric shapes are possible. Certainly, the addition of an input and output processor, i.e., 8089, would aid significantly in implementing a memory management system and file structure. Another major improvement involves the addition of a hardware floating-point coprocessor like the 8087. This speeds up computations and the turnaround for number crunching applications. Recently announced products like the 8206 memory error detection/correction unit and the 8207 dual-port dynamic RAM controller would improve fault tolerance and streamline use of random logic (SSI, MSI). Finally, the fundamental application and problem characteristics best suited to replicated shared memory structures require identification. With this known in advance, designers can quickly evaluate the merit of using a replicated shared memory structure.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of this project by the Intel Corporation and the Hewlett-Packard Company. Their donations of research and development equipment significantly contributed to the success of this research.

REFERENCES

1. Anderson, G. A. "Computer Interconnection Structures: Taxonomy, Characteristics and Examples." *ACM Computer Surveys,* 7 (1975), pp. 197–213.
2. Chang, S. S. L. "Multiple-Read Single-Write Memory and Its Applications." *IEEE Transactions on Computers,* C-29 (1980), pp. 689–694.
3. Chu, W. W., and P. B. Korff. "Multiaccess Memory: An Overview." *Proceedings of the 4th Texas Conference on Computing Systems.* Silver Spring, Md.: IEEE Computer Society Press, 1975, pp. 2B-1.1 to 2B-1.8.
4. Covo, A. A. "Analysis of Multiprocessor Control Organizations with Partial Program Memory Replication." *IEEE Transactions on Computers,* C-23, (1974), pp. 113–120.
5. Despain, A., and D. Patterson. "X-Tree a Structured Multiprocessor Computer Architecture." *Proceedings of the 5th Symposium on Computer Architecture.* Silver Spring, Md.: IEEE Computer Society Press, 1978, pp. 144–151.
6. Feng, T. "A Survey of Interconnection Networks." *Computer,* December 1981, pp. 12–27.
7. Goke, R., and G. J. Lipovski. "Banyan Networks for Partitioning on Multiprocessor Systems." *Proceedings of the 1st Symposium on Computer Architecture.* Silver Spring, Md.: IEEE Computer Society Press, 1973, pp. 21–30.
8. Haynes, L. S., et al., "A survey of highly parallel computing," *Computer,* January 1982, pp. 9–24.
9. INTELLECT-iSBC 86/12 Interface and Execution Package. No. 9800743A. Prepared by the Intel Corp., Santa Clara, California, 1979.
10. Kung, H. T. "Why Systolic Architectures?" *Computer,* January 1982, pp. 37–46.
11. Lillevik, S. L., H. T. Voorheis, and M. L. Skinner. "Multiport Memory Design." *Proceedings of the 14th Symposium on Mini and Microcomputers.* Anaheim, Calif.: ACTA Press, 1981.
12. Locanthi, B. N. "The Homogeneous Machine." Technical Report 3759, Computer Science Department, California Institute of Technology, January 1980.
13. Moore, C. "Multitasking for the 8086." Application Note AP-61, Intel Corp., Santa Clara, California, July 1979.
14. Myer, T. H. and I. E. Sutherland. "On the Design of Display Processors." *Communications of the ACM,* 11 (1968), pp. 410–414.
15. Pearce, R. C., and J. C. Majithia. "Analysis of a Shared Resource MIMD Computer Organization," *IEEE Transactions on Computing,* C-27 (1978), pp. 64–67.
16. Satyanarayanan, M. "Multiprocessing: An Annotated Bibliography." *Computer,* May 1980, pp. 101–116.
17. Skinner, M. L., and S. L. Lillevik. "Design of a Multiprocessor Computer with Multiport Shared Memory." *Proceedings of the IEEE 1982 Region 6 Conference.* Piscataway, N. J.: IEEE Press, 1982, pp. 158–164.
18. Stone, H. "Parallel processing with the perfect shuffle," *IEEE Transactions on Computing,* C-20 (1971), pp. 153–161.

Reconfigurable architectures for VLSI processing arrays

by MARIAGIOVANNA SAMI and RENATO STEFANELLI
Politechnico di Milano
Milan, Italy

ABSTRACT

Definition of architectures capable of fault tolerance and reconfiguration, suitable for very large scale integration (VLSI) implementation, is an important problem with regard to both production yield and run-time availability of VLSI devices. The case considered in the present paper concerns regular arrays of processing elements, such as the ones found in signal processing and other dedicated structures. It is proposed to achieve fault tolerance through the introduction of spare elements and reconfiguration algorithms implemented by suitable dedicated circuits and signals. A number of reconfigurable structures are presented, with different numbers and patterns of spare elements and with varying degrees of fault tolerance. Underlying fault assumptions are discussed and performances are analyzed; while architectures examined in detail consist of combinatorial elements with fairly simple interconnection schemes, extension to a wider class of structures is also considered. Implementation of diagnosis and reconfiguration is carried out at gate level: the resulting complexity is seen to be minor, as compared to the overall architecture complexity.

A. INTRODUCTION

The advent of very large scale integration (VLSI) devices introduces the problem of designing fault-tolerant and self-reconfiguring devices; this has the dual scope of allowing run-time reconfiguration after fault and initial configuration (so as to increase production yield).

The design of very reliable VLSI units has been studied both for regular structures and for complex, irregular architectures. In particular, in this last instance, the case of microprocessors has been considered.[1,2] Even though these examples refer to units capable of self-diagnosis but not of reconfiguration after fault, still they make it apparent that irregular structures require high redundancy in order to obtain significant fault tolerance. In fact, even with fairly favorable fault assumptions, it is necessary to predict the individual fault tolerance for the different units making up the device (as well as, obviously, for their interconnecting system). This introduces further problems in evaluation, for example of final yield, since the chip area becomes in turn much larger.

The case is radically different for *regular* structures, that is structures in which a number of identical cells are interconnected by means of a regular pattern. In this case, fault assumptions refer to the failure of individual cells, cell patterns, or of the interconnection substructure; thus, redundancy is considered not with respect to the whole system but with respect to a number of "spares" that can be substituted for *any* failed cell (with suitable interconnection facilities).

Memories are a basic instance of such regular structures;[3,4] yet their interest is much larger. Fast, dedicated computer architectures such as systolic arrays, mesh-connected multiprocessors, and so on,[5,6,7] and in general bidimensional signal-processing arrays belong to this class.

The problem of redundant regular structures other than memories has been considered in particular by Mangir and Avizienis,[8] who considered a rectangular array and presented a statistical analysis of fault-tolerance under various fault assumptions and spare availability.

In the present paper a rectangular array of processing cells is considered. For simplicity's sake we examine networks of memoryless cells, in which reconfiguration does not involve saving and transferring stored information or input queues.

After an initial description of the basic structure, a fault model is presented. Assumptions concerning both fault distribution and cell self-diagnosis capabilities are introduced; they are then suitably extended when spare cells are added to the basic structure and involve also additional circuits and interconnections for propagation of diagnostic signals and for self-reconfiguration.

From the basic structure some fault-tolerant ones are derived, involving different numbers and patterns of spare cells. For each of them, fault patterns that can be overcome are identified.

A balance is sought between the increase in the number of faults that can be corrected through reconfiguration (by insertion of spares) and the increase in complexity (hence chip area); in fact, above a given level added chip area and interconnection complexity lead to decreasing reliability and yield.[8]

For the above structures, reconfiguration algorithms are presented; it is proved that they lead to reasonably simple interconnection structures even after reconfiguration. In particular, the length of interconnections between reconfigured cells is proved to be quite limited even in the worst case. For each algorithm, reliability of the complete structure and other statistical informations are evaluated.

Finally, we consider some extensions of the present proposal to structures with sequential cells and/or different interconnection patterns.

B. BASIC STRUCTURE

The basic structure considered here is a rectangular array of memoryless cells (processing cells), interconnected by a regular pattern of connections along the two Cartesian axes; information flows in one direction only for each axis, namely left

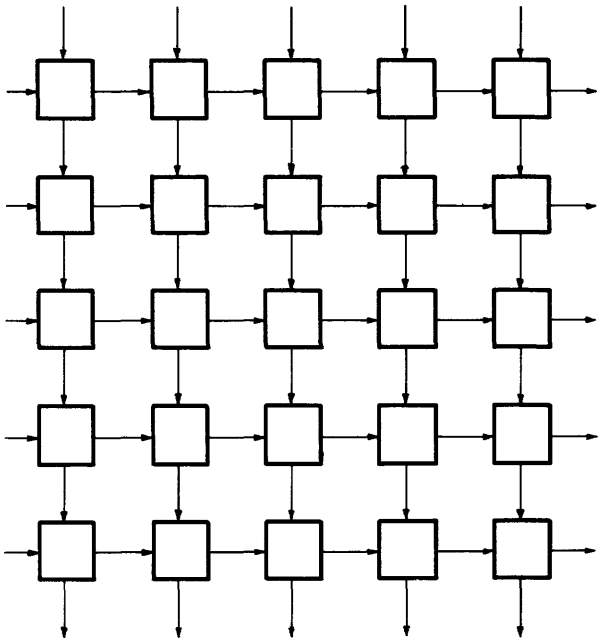


Figure 1—Basic structure

to right along the horizontal one and down along the vertical one (see Figure 1). This assumption allows greater simplicity in designing the cell and additional circuits for fault tolerance; extension to any regular interconnection structure is quite easy.

Each cell is numbered according to a matrix ordering; the cell on row i, column j is designated as cell (i,j). Numbering indices are said to be the *physical indices* of each cell, denoting its physical position in the array. Although the assumption is not essential, we assume the matrix to be a square one so that cell numbers go from (1,1) to (N,N). Interconnections are oriented along increasing values of i and j. Cell (i,j) receives information from two adjacent cells, namely $(i-1, j)$ (providing the vertical input) and $(i, j-1)$ (providing the horizontal input) and gives information to $(i, j+1)$ (horizontal output) and $(i+1, j)$ (vertical output). Links between any pair of adjacent cells are assumed to have unit length. Border cells, i or j taking on values either 1 or N, receive part of their input from, or give part of their outputs to, interconnections with external systems. The structure is then strictly iterative.

From this basic structure, allowing no reconfiguration after failure, fault tolerant ones are derived by adding suitable patterns of spare cells (cells that are not active if the cells of the basic structure are all in a working state) and additional interconnection links. Though in the basic structure all interconnections are always active, in the fault-tolerant ones it is necessary to insert control circuits along the interconnection paths so that only the correct links will be active, and only in the scheme defined by the reconfiguration algorithm.

C. FAULT MODEL

Like all fault-tolerant computing design the design self-repairing VLSI structures requires the formulation of suitable fault assumptions that allow one to reach, for each given structure, an acceptable and realistic compromise between added complexity and probable resistance to a reasonable spectrum of faults.

In the present case, we do not make any assumption concerning *elementary* faults (i.e., faults at elementary logic circuit level, e.g., gate level); in fact, where VLSI devices are concerned, it is in practice impossible to find physical- or even logical-level failure assumptions leading to models at once realistic and manageable.

We prefer to consider as basic the processing cells making up the array, which are already fairly complex elements in the architectures here considered. A fault in this cell can best be defined as "incapacity of correct operation, given correct inputs," possibly caused by a set of physical or gate-level faults.

The cell is assumed to be *capable of self-diagnosis* to the point of giving correct information on the existence of internal faults that lead to incorrect operation. Even a faulty cell is assumed to generate correct error information. In fact, the cell could be capable of internal gate level fault tolerance and reconfiguration; in this case, the failure signal would be made active only when internal reconfiguration limits are exceeded.

A basic fault assumption here adopted is *mutual independence of cells* where failure is concerned; a fault in cell (i, j) does not in any way postulate failure of any other (adjacent) cell. This assumption can be reasonably justified by observing that the basic structure here considered leads to layouts where densely populated chip areas, the processing cells, are separated by areas reserved to the simpler and less dense interconnection circuits. Obviously, such macroscopic failure causes as mechanical ones, leading to massive destruction of large chip areas, are not considered here.

Interconnections will at first, in the initial analyses be considered to be always correctly working, which allows us to introduce reconfiguration algorithms and control circuits with greater simplicity. Later, link failures also will be considered. In parallel with our assumptions for cell self-diagnosis, we assume that the presence of failure on links can always be detected, owing to the adoption of suitable error-detecting codes for all information transfer.

Transition from a basic structure to fault-tolerant ones involves the introduction of spare cells and of additional circuits for error signal propagation and for reconfiguration purposes supporting each processing cell. Assumptions concerning the performance of these circuits will be discussed in Section D.1.

A further assumption is that the chip area due to such additional circuits, as well as to additional interconnections allowing reconfiguration after fault, is much smaller (at least by one order of magnitude) than the area due to the basic structure. This allows us to make some basic simplifications in reliability calculations, and it is well justified in view of the application class envisioned for our architectures and of the ensuing complexity of the processing cells.

As already stated, the architectures here presented can be adopted for run-time reconfigurable structures, as well as for initial configuration at production certification time. Although in the case of initial configuration all faulty cells can be detected by external instruments and unnecessary links can be deleted, for example by "blowing" suitable fuses, in the case of run-time reconfiguration the link control must be performed by specific internal circuits. It then becomes necessary to introduce additional assumptions concerning the relative occurrence of cell failures in time. We assume that cell failures occur in sequence, that is that no two simultaneous cell failures ever occur. This allows correct reconfiguration of the system after each fault, or else a correct generation of the worst case exceeded signal, without the necessity of unduly complex additional structures. The propagation time for error signals in combinatorial architectures is assumed to be very small compared to processing times.

In the present paper, particular attention is given to run-time reconfiguration.

In the next section, we present for various structures the curves detailing the probability of reconfiguration to working state after increasing numbers of independent errors, both with increasing number of cells and with increasing complexity of spare configuration.

D. FAULT-TOLERANT ARCHITECTURES

In this section, structures allowing tolerance of various numbers and patterns of faults will be considered. Architectures will be examined in detail, both in their general reconfiguration policies and in their additional circuits designed for this purpose.

All structures are derived from the basic (N*N) array by insertion of various patterns of spare cells. Spares are always organized in regular patterns; this lets us keep a totally repetitive design for interconnections and for cell circuits (in particular, for reconfiguration circuits).

Starting with a totally correct structure, reconfiguration after fault and consequent deformation of interconnection patterns will be analyzed. Although obviously the whole set of allowable interconnections is present in the structure, the working pattern at any given time will be activated by error and reconfiguration signals present in the network.

The insertion of k spare cells could in principle allow reconfiguration after any pattern of up to k cell failures, provided each spare were connected with any other cell in the structure. Yet even for a very small number of cells this would lead to unacceptable complexity of interconnection links and of controlling circuits. We prefer here to limit the set of fault patterns to which the reconfiguration applies, so as to guarantee reasonable simplicity of resulting interconnections and circuits.

This procedure allows us always to see reconfiguration policies in a global way, involving each time the whole structure.

Given the initially working array of N*N processing cells described in Section B, the response to a failure involves exclusion of the faulty cell, insertion into the working set of a suitably chosen cell, and total reconfiguration of interconnections. To this purpose, working cells receive now *logical* indices (i', j') that may be different from the physical ones, and interconnections are activated between cells having *logical* indices at unit distance. Reconfiguration, then, involves a global renaming procedure; this procedure is started whenever a cell fails. The algorithms here presented aim at reducing the complexity of interconnections between cells that might be termed *logical neighbors*. Circuits controlling link activation may be considered as computing logical indices for each cell, starting from physical indices and error information. This allows us to consider *any* cell (even spare ones) as having a standard set of links and control circuitry, leading to potential interconnection with a small number of neighboring cells.

Failure of a link between two cells will be assimilated to failure of the cell from which link originates; this will minimize the added complexity in reconfiguration circuits.

Such policies still allow us to cover a relevant number of fault instances. Actually, fault patterns excluded by all of the reconfiguration criteria here presented are the ones in which many faulty cells are clustered together in a restricted area, an instance mostly created by mechanical or other major manufacturing problems. It seems reasonable to exclude devices so characterized from acceptance tests. The following reconfigurable architectures will be considered:

1. N spare cells, organized in one column (structure no. 1)
2. 2N spare cells, organized in
   a. two columns (structure no. 2)
   b. one column, one row (structure no. 3)

Each instance will be considered in detail, analyzing its performances and the additional circuits it requires in order to perform correct reconfiguration. Some common assumptions are made, namely

- A spare cell, when not in use, is denoted by logical index 0 on the main reconfiguration axis.
- A faulty cell, after reconfiguration, is characterized by logical index 0.
- A faulty cell may still allow passage of correct information links; more correctly, its link control circuits may still work properly and allow it to complete an interconnection.

Therefore, the possibility of link failure must be considered separately in the case of a "passing" link and in the case of other links. The first instance inevitably leads to fatal failure (on the other hand, failure of a cell and its passing link corresponds to the fault clustering that we do not allow for our reconfiguration). The second instance, on the contrary, can be assimilated to cell failure and suitably analyzed.

D.1 Structure with N Spare Cells

Given an array structure of N*N processing cells interconnected via a rectangular grid, the simplest regular pattern of spare cells consists in a column (row) of N cells, added on one arbitrarily chosen side. Here we consider a structure of type no. 1, as shown in Figure 2.

The resulting physical structure consists then of $N*(N + 1)$ cells. We define the reconfiguration rule by stating that reconfiguration may take place only along the axis of information flow leading to the spare cells—here, along rows only. As a consequence, up to N cell failures may be corrected provided

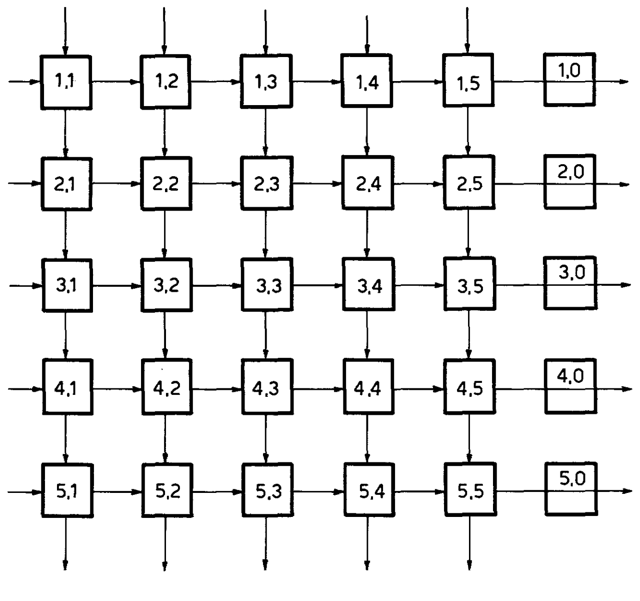


Figure 2—Structure no. 1 with no faults, spare column on the right. Each cell is marked by its original pair of logical indices.

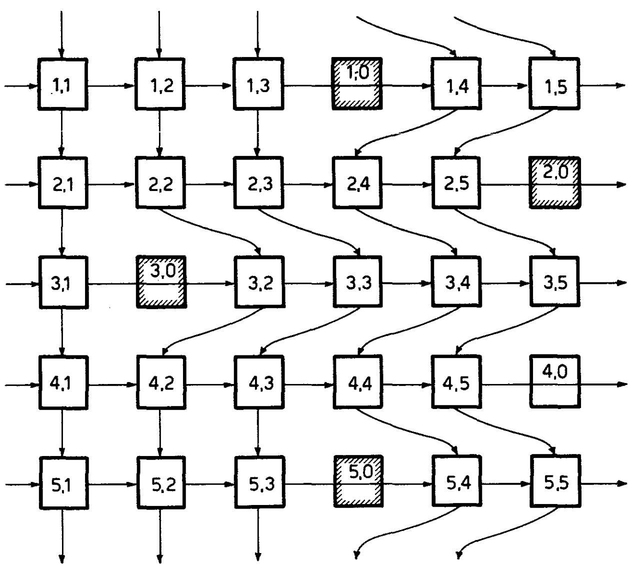


Figure 3—Structure of Figure 1 after faults and reconfigurations. Each cell is marked with new logical indices.

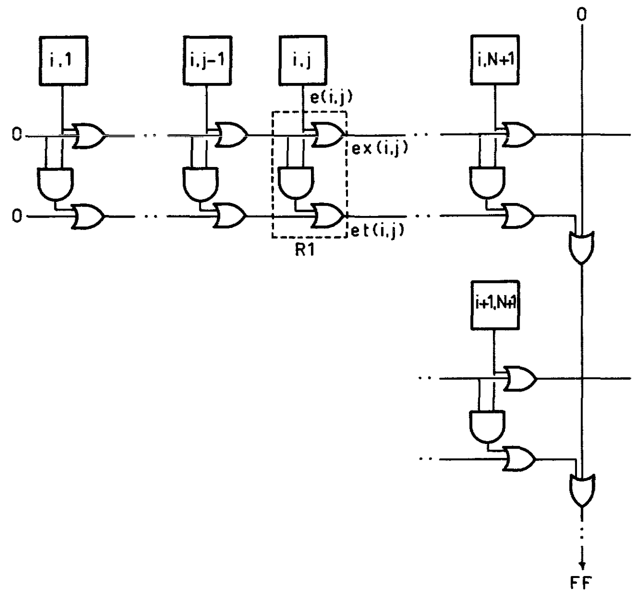


Figure 4—Creation of deformation (ex) and fatal failure (et) signals for structure no. 1

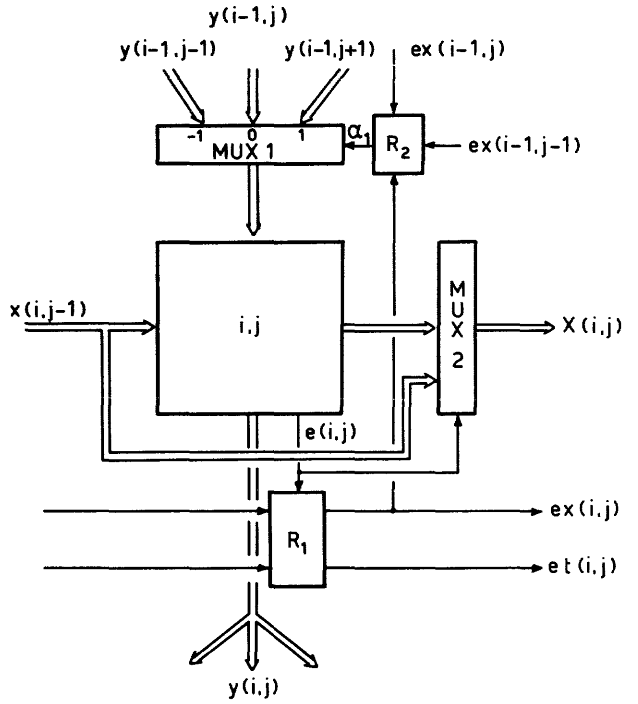


Figure 5—Basic cell structure

no two failures occur in the same row; this is the only restriction to fault patterns allowed. Assuming then that physical cell (i, j) fails, its functions are thereafter performed by physical cell $(i, j+1)$, which now gets logical indices $i' = i$, $j' = j + 1$—for higher values of j, it is $j' = j + 1$, $i' = i$. The failed cell gets $j' = 0$; for any cell (i, k), $k < j$, it is $k' = k$.

Consider first the assumptions restricting failures to cells. The cell reconfiguration and renaming procedures are very simple; an example is given in Figure 3. After renaming, links will be activated to interconnect cells with correctly ordered *logical* indices. Traversal of faulty cells by working links follows the direction of reconfiguration, that is, along the rows.

Given any pair of cells initially connected by a direct link, after any acceptable failure and reconfiguration they will reconfigure as two cells connected by a link that is one unit longer than the original one.

All of these considerations lead to design of circuits performing

1. Error signal propagation. To this end, two pieces of information must be inserted: the first one points out to the cells from (i, j) to (i, N) the necessity of reconfiguration along row i; the second one points out the existence of a *fatal failure*, that is, overflow of worst case allowed.
2. Link control. This is to achieve correct interconnections. (See the transformation from Figure 2 to Figure 3).

For error signal propagation, we introduce the following signals (see Figure 4):

- $e(i, j)$: created by the cell, with value 0 if the cell is working, 1 if it is faulty
- $ex(i, j)$: propagated along row i with value 1 if any of the cells from $(i, 1)$ to (i, j) is faulty
- $et(i, j)$: propagated along row i with value 1 if two or more cells from $(i, 1)$ to (i, j) are faulty.

A final column of *OR* gates generates fatal failure information.

To consider link control, refer to Figure 3; any cell (i, j) may receive its vertical input from any one of the cells $(i-1, j-1)$, $(i-1, j)$, $(i-1, j+1)$, depending on the fault pattern in rows $i-1$ and i, and may either forward the information it has processed (if it is itself correctly working) or let the information processed by $(i, j-1)$ pass (in case it has failed). Reconfiguration rules do not allow any other instance.

These alternatives are implemented by inserting on any cell two multiplexers (see Figure 5): one at the horizontal output of cell (i, j), the other one at its vertical input. In the horizontal multiplexer, internal error message $e(i, j)$ selects either processed information $(e(i, j) = 0)$ or the passing information $(e(i, j) = 1)$; in the vertical multiplexer, a three-valued code a selects information coming from cells $(i-1, j-1)$ (value -1), $(i-1, j)$ (value 0), $(i-1, j+1)$ (value $+1$), as shown in Table I.

TABLE I—Values of a for structure no. 1 (ex$(i, j) = 1$ for one faulty cell between $(i, 1)$ and (i, j), 0 for no faulty cells)

ex$(i,j) = 0$,	ex$(i\text{-}1,j\ \ ) = 1$,	$a = 1$
ex$(i,j) = 1$,	ex$(i\text{-}1,j-1) = 1$,	$a = 0$
ex$(i,j) = 0$,	ex$(i\text{-}1,j\ \ ) = 0$,	$a = 0$
ex$(i,j) = 1$,	ex$(i\text{-}1,j-1) = 0$,	$a = -1$

Considering then the column of spare cells, a spare will let information pass through without any processing if it is $e(i, N+1) + ex(i, N+1) = 1$. Moreover, correct operation requires a final row of multiplexers below row N, so as to allow correct outputs.

It may be noticed that although failed cells also receive input information, their outputs are not forwarded; while horizontal output is blocked by the output multiplexer (with signal $e(i, j) = 1$) values for a in adjacent cells guarantee that the error will never be propagated.

It is assumed that all error or error-related signals are always correct (if necessary, coding techniques might be adapted to this purpose); the vertical input multiplexer must be self-checking so as to correctly contribute to creation of error signals; all horizontal input multiplexers must always be correctly working (possibly doubled up for this purpose) even in faulty cells.

In order to evaluate systems' performances (and compare them with those for other architectures) a number of statistical computations concerning the reconfigurable structure are now discussed.

In Figure 6, the probability of system survival through reconfiguration is plotted against number of cell failures for various values of N. Probability of correction for the first failure is obviously 1. On the other hand, the probability of reaching fatal error conditions even after inserting a small percentage of faulty cells is not negligible. This fact is made more evident by the degree of spares utilization, which is defined as the number of failures that can be overcome with a given probability against the number N of spares (therefore, once again, related to the dimensions of the array). In Figure 7 three curves are plotted, corresponding to the probabilities 25%, 50%, and 75%. The results obtained are not very satisfactory; it can be seen that in large arrays only a small percentage of spares are effectively used. This consideration would lead us to prefer smaller arrays, but on the other hand in small arrays the relative increase of chip dimensions due to the insertion of spares becomes a relevant factor—the increase coefficient is $(N+1)/N$—and its influence on yield cannot be overlooked.

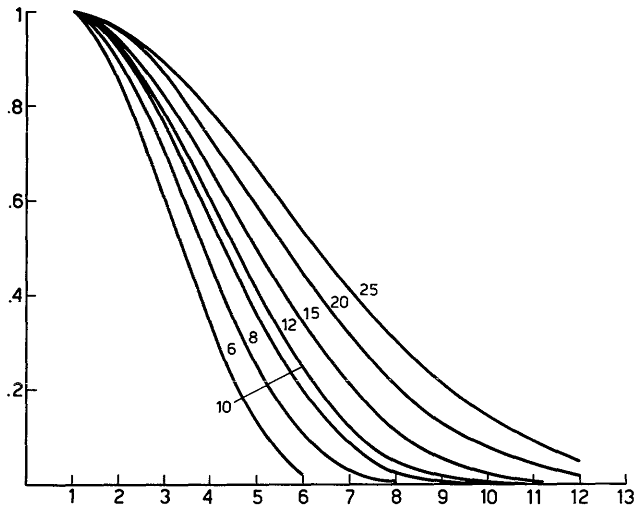


Figure 6—Probability of system survival against number of faults for structure no. 1

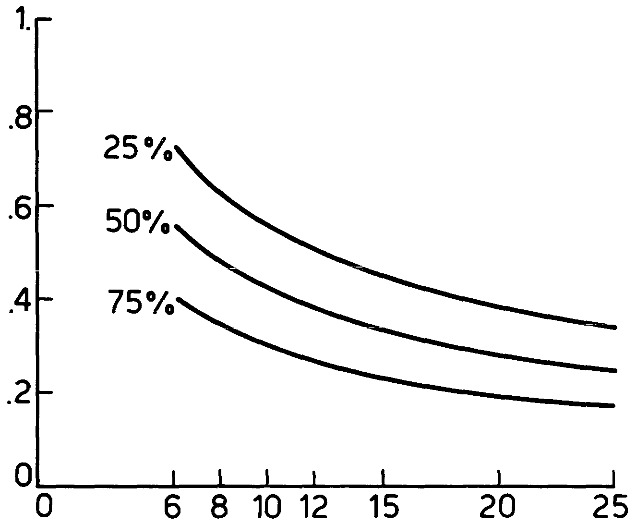


Figure 7—Degree of spares utilization against dimensions of basic structure for structure no. 1

Consider now the possibility of link failure. If a link fails, this amounts to unavailability of information produced by the cell from which the link originates; it may, then, be reasonably assimilated to failure of this same cell. It becomes now necessary to introduce in the input multiplexers of (i, j) error-detecting circuits producing error signals that will be OR-ed

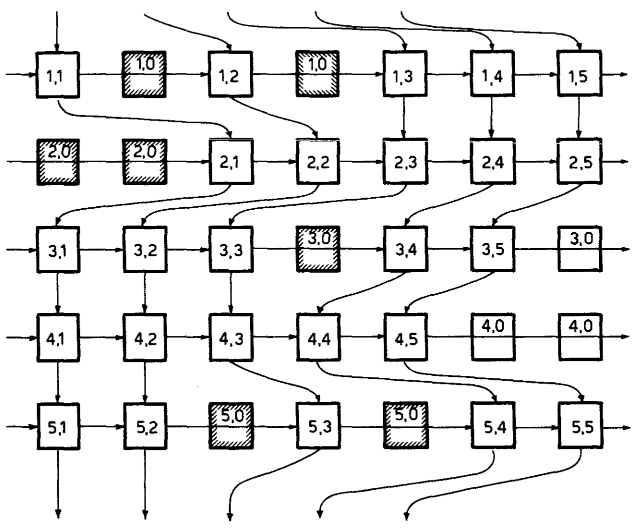


Figure 8—Structure no. 2 with two added columns on the right side

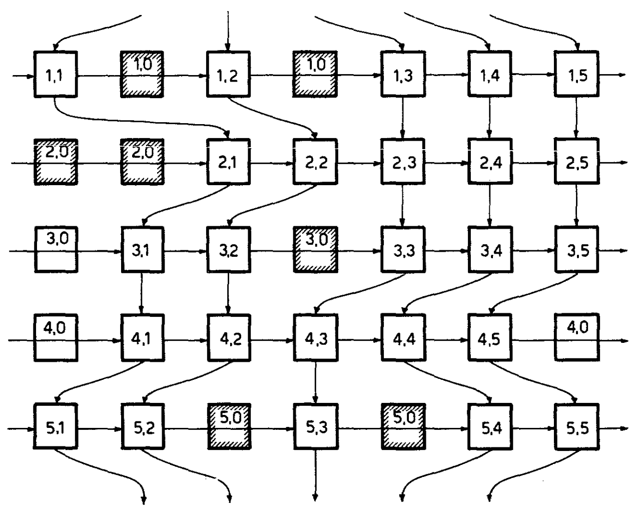


Figure 9—Structure no. 2 with one added column on each side

with the error signal of the originating cell. This same philosophy can be adapted also for all subsequent architectures.

*D.2. Structures with 2*N* Spares

Two alternative solutions are considered: spares are added along two columns (rows) or else along one column and one row. Resulting circuits for reconfiguration and error propagation are different, as are reconfiguration rules and error signals. The two cases are therefore examined separately.

D.2.1. Structure with two added columns. In order to keep the regularity of cell and interconnection structures, columns are added at the sides of the array: either one spare column on each side or both on one side of the basic array.

These are shown in Figures 8 and 9: both show structures involving reconfiguration after faults. Error signals required and statistical results are identical for both; on the other hand, circuits for error detection, error signal propagation, and reconfiguration are different (circuits for the two instances are shown in Figure 10). We consider here in detail the case of one column on each side.

Again, reconfiguration takes place only along one axis, in the direction of the two added columns. It becomes possible, then, to correct up to two faults in each row; no other restriction for fault patterns is introduced. In order to obtain

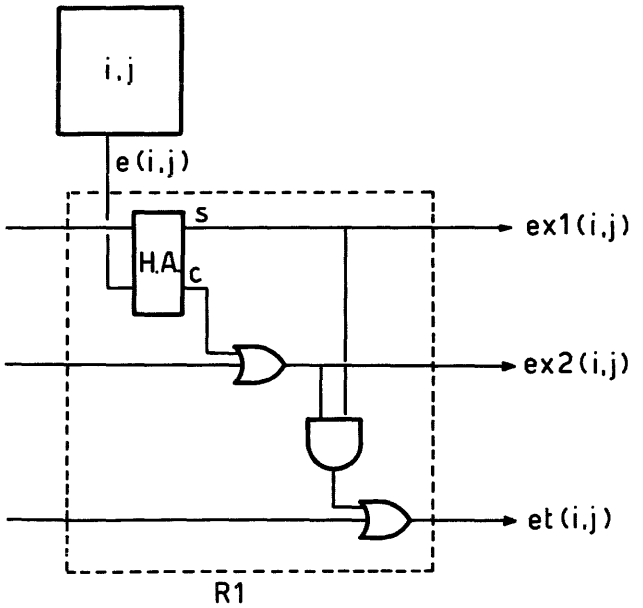


Figure 10—Creation of deformation (ex1, ex2) and fatal failure (et) signals for structure no. 2

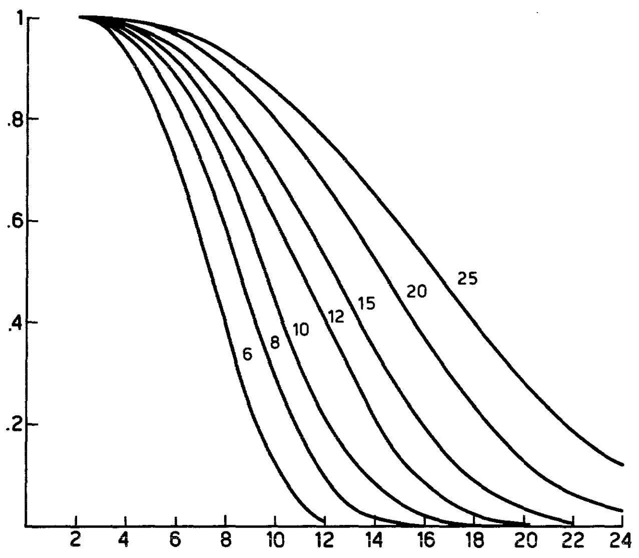


Figure 11—Probability of system survival against number of faults for structure no. 2

correct reconfiguration, two error signals must now be propagated through each row of the array (see Figure 10), specifying the three possible instances: no error to the left of jth cell ($ex1 = 0$, $ex2 = 0$), one error ($ex1 = 1$, $ex2 = 0$), two errors ($ex1 = 0$, $ex2 = 1$). As in the case with one column of spares, each cell gives one failure signal $e(i, j)$. Fatal failure signal $et(i, j)$ denotes the presence of more than two errors in a row.

Each cell in a row may now receive its vertical input from *five* different sources compared to three in the first structure); that is, (i, j) may receive vertical inputs from any of the five cells from $(i-1, j-2)$ to $(i-1, j+2)$. Figure 9 shows all possible instances.

The structure of the processing cell is then modified, compared to the case of one column of spares, only by replacing the three-way multiplexer of Figure 5 with a five-way multiplexer and, obviously, by suitably modifying information a. Table II gives the values of a for structure no. 2, where by $er(i, j)$ we denote the number of errors coded by signals $ex1(i, j)$ and $ex2(i, j)$ in Figure 10.

Circuit R2 in Figure 5 shall therefore receive, as its inputs, all signals listed in Table II.

TABLE II—Values of a for structure no. 2 (er(i, j) is the number of errors coded by signals ex1(i, j) and ex2(i, j))

er$(i,j) = 2$,	er$(i-1, j-2) = 0$,		$a = -2$
er$(i,j) = 1$,	er$(i-1, j-1) = 0$,		$a = -1$
er$(i,j) = 2$,	er$(i-1, j-1) = 1$,	e$(i-1, j-1) = 0$	$a = -1$
er$(i,j) = 0$,	er$(i-1, j) = 0$,		$a = 0$
er$(i,j) = 1$,	er$(i-1, j-1) = 1$,	e$(i,j) = 0$	$a = 0$
er$(i,j) = 2$,	er$(i-1, j-1) = 2$,		$a = 0$
er$(i,j) = 0$,	er$(i-1, j) = 1$,	e$(i-1, j+1) = 0$	$a = +1$
er$(i,j) = 1$,	er$(i-1, j) = 2$,		$a = +1$
er$(i,j) = 0$,	er$(i-1, j+1) = 2$,		$a = +2$

Statistical results concerning probability of survival after faults are plotted in Figure 11. It can be seen that improvement with respect to structure no. 1 is greater than can be attributed simply to doubling the number of spares; this can also be seen in Figure 12, where spare cell utilization is plotted. Such improvements are balanced, of course, by greater complexity of added circuits and signals.

D.2.2. Structure with one row, one column added. Here again, the choice of edges to which spares are added is not essential to the results achieved; we choose to add one row and one column in the direction of information flow.

Assumptions concerning fault distribution and reconfiguration rules become more complex; in fact, reconfiguring along two axes may lead to conflicts between possible alternatives. Whereas in all previous structures only index j' had to be computed after reconfiguration, now the renaming procedure assigns to any cell a *pair* of logical indices corresponding to reconfiguration after fault along either axis. To this

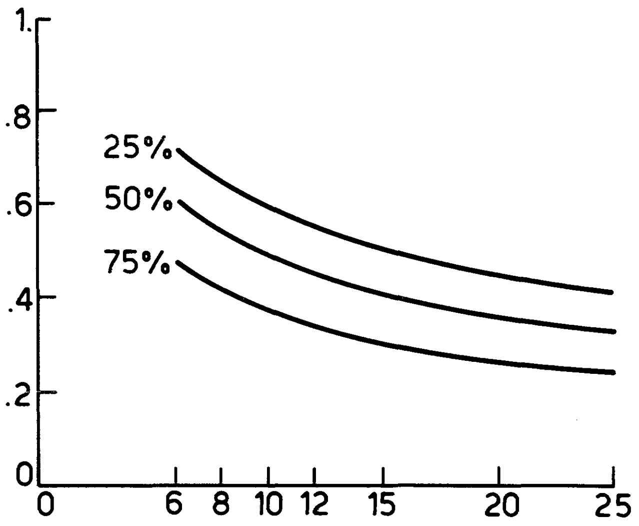


Figure 12—Degree of spares utilization against dimensions of basic structure for structure no. 2

purpose, it is necessary to introduce a policy assigning to any failed cell a direction (vertical or horizontal) of reconfiguration: we refer from now on, correspondingly, to *vertical* or *horizontal* faults.

This policy involves the intrinsic limitation (consistent with the ones adopted for previous structures) that no row or column may have more than one failed cell in the direction of reconfiguration. Some of the rather complex clusters of faulty cells that this limitation would still allow would unfortunately require very complex error information and redundant interconnection links. We choose, therefore, to introduce a further restriction:

Let (i, j) be a vertical fault: then no other cell (k, j) with $k > i$ is faulty.

Reconfiguration and renaming rules may then be summarized as follows:

- Scan each column upwards (i.e., for decreasing values of i from N to 1); as soon as a faulty cell is identified, it is marked as a vertical fault.
- Classify all other possible faulty cells as horizontal faults.
- If no row has more than one horizontal fault, reconfiguration is then possible.

These rules are implemented by means of the circuit in Figure 13. Signal ec propagates upwards and gets value 1 from the lowest faulty cell up. The signal vf (vertical fault) is 1 only for the lowest faulty cell in the column, according to our restriction. The signal ey propagates to lower cells the information on vertical deformation.

All faulty cells in a column, above the lowest one, get signal hf (horizontal fault) = 1; hf propagates to the right, by means of signal ex, so as to denote horizontal deformation. Two (or more) horizontal faults in a row cause the signal et (fatal failure) to get value 1.

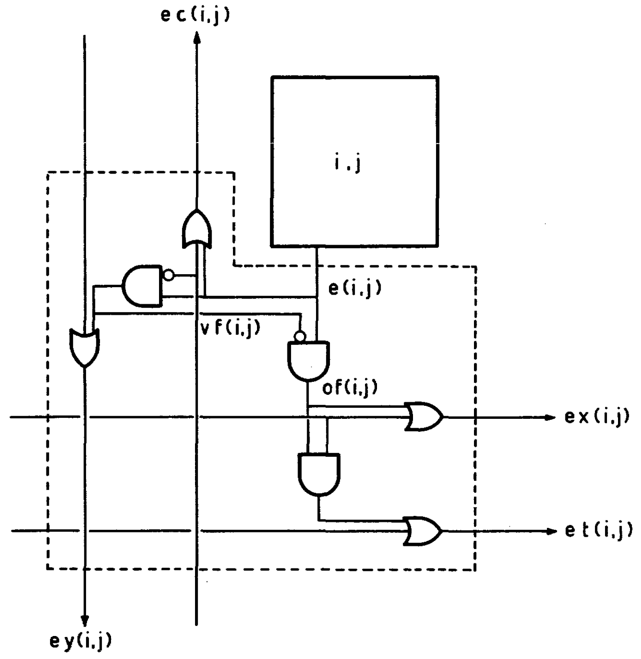


Figure 13—Creation of diagnostic signals for first failure (ec), vertical deformation (ey), horizontal deformation (ex), and fatal failure (et) for structure no. 3

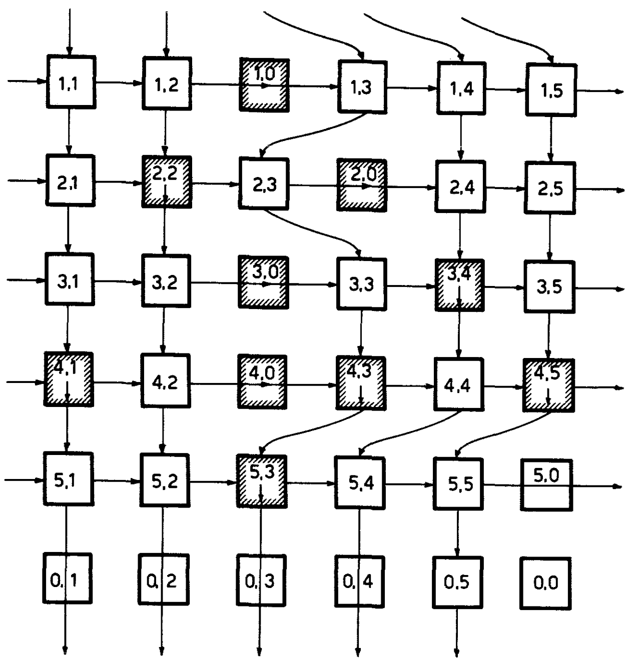


Figure 15—First sample structure: first (horizontal) deformation

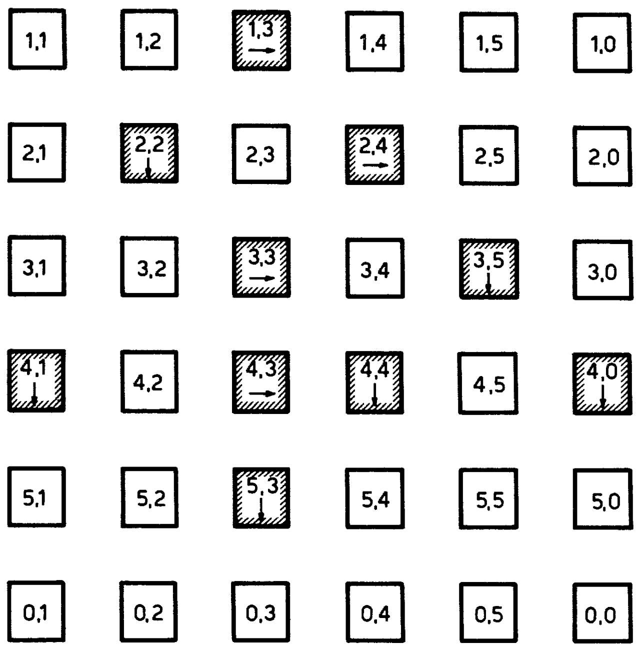


Figure 14—Vertical and horizontal fault classification

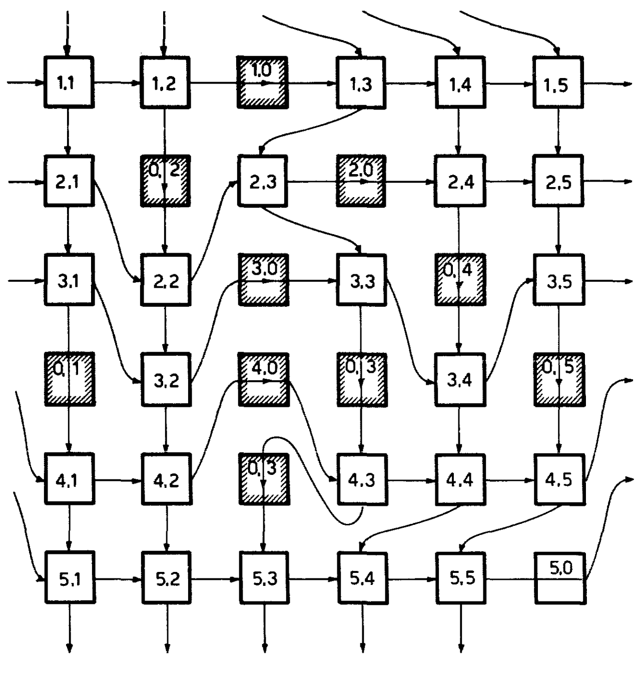


Figure 16—First sample structure: second (vertical) deformation

The procedure has been applied to the example in Figure 14, where each cell is marked with its *physical* indices and (if faulty) with the chosen reconfiguration direction.

For greater clarity, we perform the reconfiguration in two logical steps (actually, both are performed at the same time by combinatorial circuits). Renaming and interconnection modification are first performed along the horizontal axis (Figure 15) and temporary logical indices i', j' are found; then the renaming and modification are performed along the vertical axis (Figure 16) and final values of logical indices i'', j'' are computed.

Figures 17 and 18 represent the effects of the first (horizon-

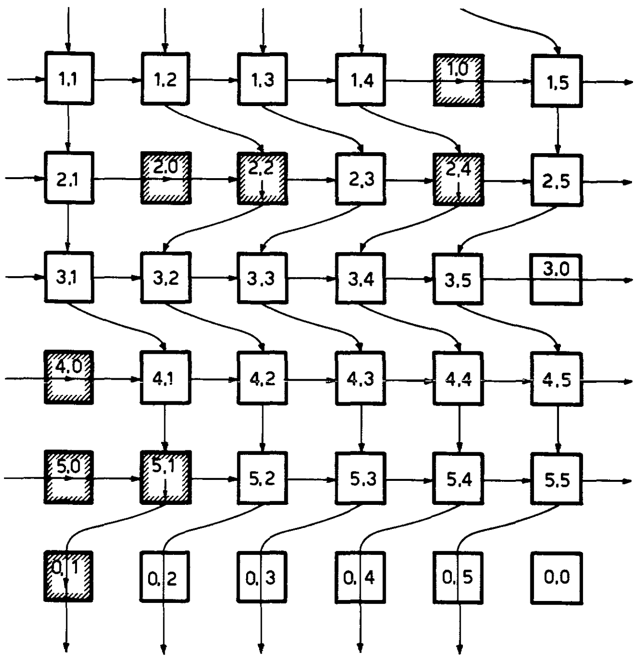


Figure 17—Second sample structure: first (horizontal) deformation

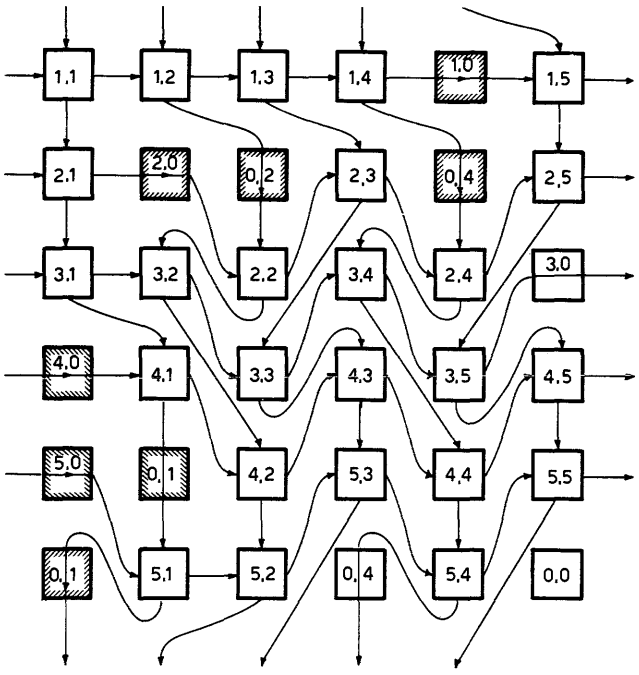


Figure 18—Second sample structure: second (vertical) deformation

tal) and second (vertical) deformations, respectively, for a fault pattern more meaningful than the previous one.

Reconfiguration obtained by two logical steps leads to the following considerations:

- Both deformations are of the same kind as the one produced by structure no. 1; therefore horizontal (vertical) faults must allow a passing link for horizontal (vertical signals.
- Links between logical neighbours have, in the worst case, length three both in structure no. 2 and in no. 3, since each simple deformation increases the maximum link length by one unit.
- Rules for structure no.3 guarantee that all pairs of logical indices i'', j'' ($1 \le i'' \le N$ and $1 \le j'' \le N$) are assigned to nonfaulty cells.

  In fact, after the first horizontal deformation is performed, all logical pairs i', j' ($1 \le i' \le N$ and $1 \le j' \le N$) are assigned either to correct cells or to vertical faults (all horizontal faults have logical index $j' = 0$). After the second (vertical) deformation, all logical pairs i'', j'' ($1 \le i'' \le N$ and $1 \le j'' \le N$) are assigned to correct cells, since a vertical fault is always the lowest faulty cell in a column.

The complex example in Figure 18 shows all possible interconnections.

Reconfiguration rules—as defined above—are not strictly isomorphic; as a consequence, possible reconfiguration instances are not equal along the two axes. In particular, while for any (i, j) vertical inputs may come from any one of cells $(i-2, j-1)$, $(i-1, j-1)$, $(i, j-1)$, $(i-1, j)$, $(i-2, j+1)$,

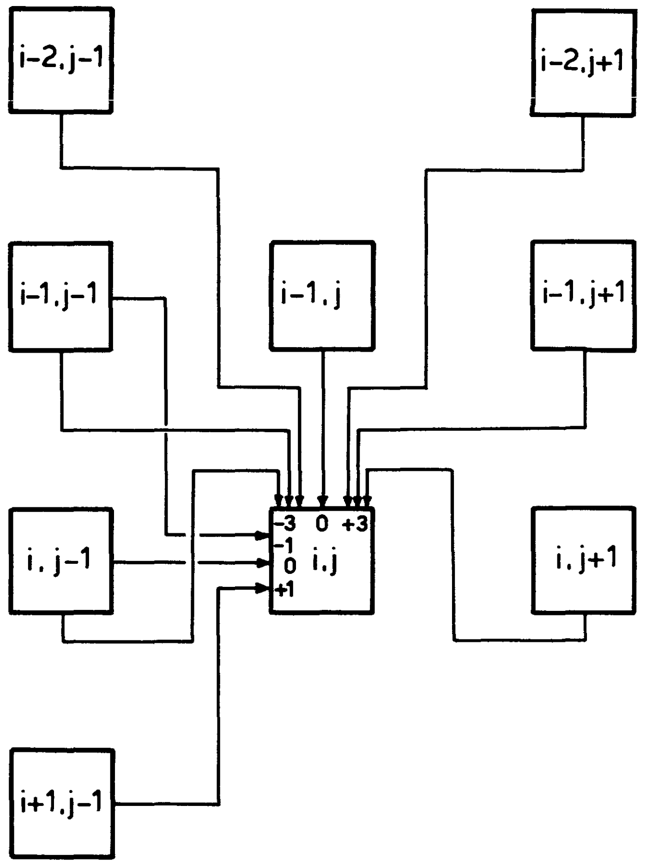


Figure 19—Possible logical neighbors of cell (i, j) after structure no. 3 reconfigurations

TABLE III—Values of a and b for structure no. 3 (ex indicates horizontal deformation; ey indicates vertical deformation)

$ex(i-1,j-1)=0$,	$ex(i\ ,j-1)=1$,	$ey(i-1,j-1)=1$,	$ey(i-1,j\ )=0$,	$a=-3$
$ex(i-1,j-1)=0$,	$ex(i\ ,j-1)=1$,	$ey(i-1,j-1)=0$,	$ey(i-1,j\ )=0$,	$a=-2$
$ex(i-2,j-1)=0$,	$ex(i-1,j-1)=1$,	$ey(i-2,j-1)=1$,	$ey(i-1,j-1)=1$,	$a=-2$
$ex(i-2,j-1)=0$,	$ex(i-1,j-1)=1$,	$ey(i-2,j-1)=0$,	$ey(i-1,j-1)=1$,	$a=-1$
$ex(i-1,j\ )=0$,	$ex(i\ ,j\ )=0$,	$ey(i-1,j\ )=0$,	,	$a=\ 0$
$ex(i-1,j\ )=0$,	$ex(i\ ,j\ )=0$,	$ey(i-2,j\ )=1$,	,	$a=\ 0$
$ex(i-1,j-1)=1$,	$ex(i\ ,j-1)=1$,	$ey(i-1,j\ )=0$,	,	$a=\ 0$
$ex(i-2,j-1)=1$,	$ex(i-1,j-1)=1$,	$ey(i-2,j\ )=1$,	,	$a=\ 0$
$ex(i-2,j\ )=1$,	$ex(i-1,j\ )=0$,	$ey(i-1,j\ )=1$,	$ey(i-2,j+1)=0$	$a=+1$
$ex(i-1,j\ )=1$,	$ex(i\ ,j\ )=0$,	$ey(i-1,j\ )=0$,	$ey(i-1,j+1)=0$	$a=+2$
$ex(i-2,j\ )=1$,	$ex(i-1,j\ )=0$,	$ey(i-1,j\ )=1$,	$ey(i-1,j+1)=1$,	$a=+2$
$ex(i-1,j\ )=1$,	$ex(i\ ,j\ )=0$,	$ey(i-1,j\ )=0$,	$ey(i-1,j+1)=1$,	$a=+3$

$ey(i-1,j-1)=0$,	$ey(i-1,j\ )=1$,	$b=-1$
$ey(i-1,j-1)=0$,	$ey(i-1,j\ )=0$,	$b=\ 0$
$ey(i-1,j-1)=1$,	$ey(i-1,j\ )=1$,	$b=\ 0$
$ey(i\ ,j-1)=1$,	$ey(i-1,j\ )=0$,	$b=+1$

$(i-1, j+1)$, and $(i, j+1)$, horizontal inputs may come only from cells $(i-1, j-1)$, $(i, j-1)$, or $(i+1, j-1)$. (See Figure 19).

The resulting cell structure is given in Figure 20. Values for codes a and b, are given in Table III.

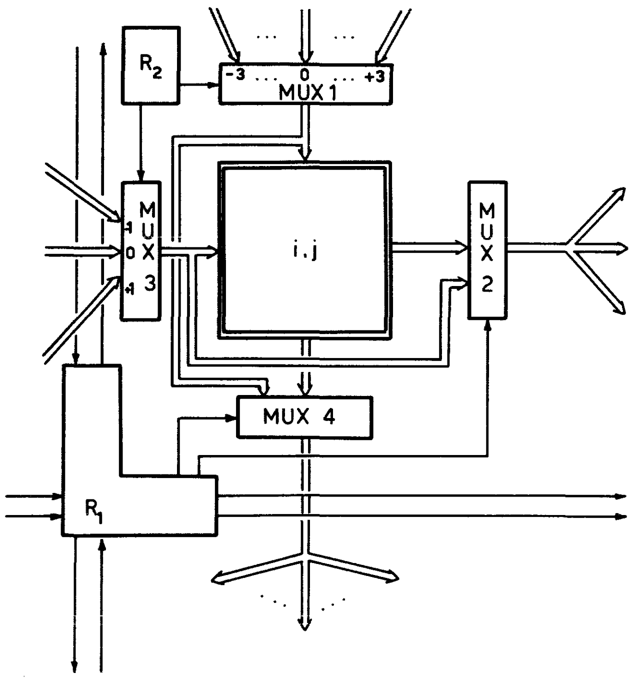


Figure 20—Basic cell structure for structure no. 3

Statistical results are computed and given in Figures 21 and 22. Compared to structure no. 2, which has almost the same number of spares, structure no. 3 achieves a noticeable improvement in both survival probability and spares utilization.

It can be seen that up to three errors are always corrected by this structure, and that four errors lead to fatal failure only when the faulty cells are on the vertices of a rectangle (that is, (i, j), (i, k), (l, j), (l, k)). The probability of such fault distribution, particularly in large arrays, is very low.

Structure no. 3's advantages over structure no. 2 are paid for by a somewhat higher complexity of added circuits and interconnections.

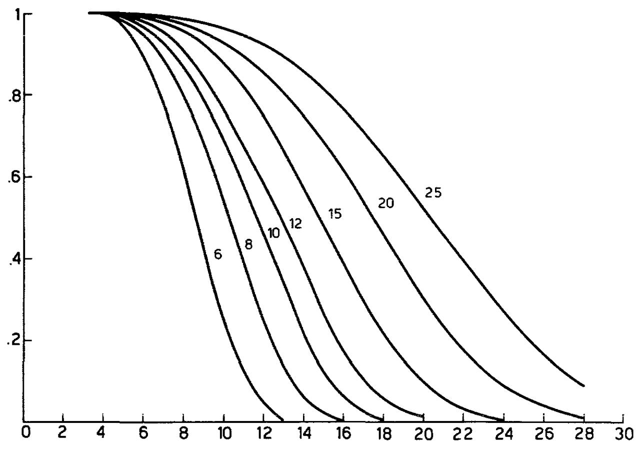


Figure 21—Probability of system survival against number of faults for structure no. 3

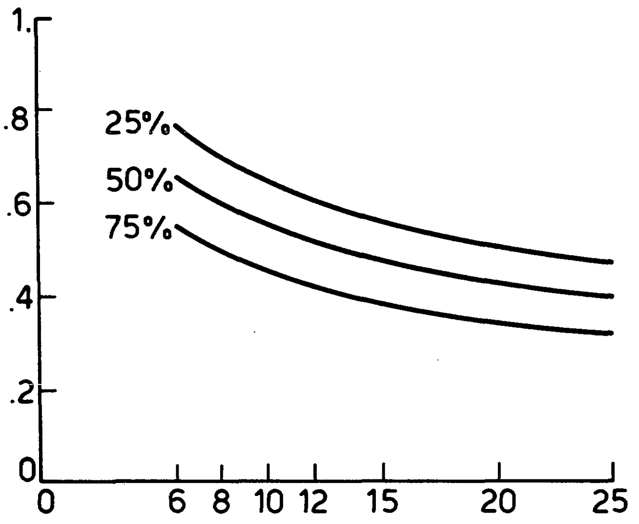


Figure 22—Degree of spares utilization against dimensions of basic structure for structure no. 3

It is evident that other, more complex structures may be examined. Actually, some particularly relevant ones (e.g., with two rows, two columns added) are being examined now.

E. CONCLUDING REMARKS

The problem of fault-tolerant, reconfigurable VLSI processing arrays has been considered.

Fault-tolerance is achieved through the introduction of spare elements and the definition of reconfiguration algorithms; the choice of spare patterns and of reconfiguration algorithms is guided by the aims of keeping

1. Complete repetitivity of processing elements and supporting circuits (i.e., so that even the reconfigurable array is iterative)
2. Reasonable simplicity of the added circuits and interconnections implementing the reconfiguration algorithm (so that chip area overhead is nearly all due to spare elements only)
3. Moderate increase of interconnection lengths, even in worst-case reconfiguration (this allows us to disregard propagation time differences between basic and reconfigured structures).

Structures proposed are characterized by increasing complexity and—at the same time—increasing probability of survival and efficiency of spares utilization. It is now possible to make some comments about extensions of the present criteria to instances not considered; the most relevant extensions seem to be the ones to (1) the interconnection structure (2) the nature of processing cells, and (3) the fault model.

1. The first extension would allow bi-directional information flow along each axis. The only relevant changes to the algorithms provide bi-directional propagation of error information along each axis and insertion of input and output multiplexers for each flow direction.

   If more complex but still regular interconnection structure are considered, as in the case of some systolic arrays, it becomes necessary to identify a larger number of alternatives for reconfiguration directions and corresponding sets of error signals.
2. Although we have considered reconfiguration to be a one-step procedure for combinatorial cells, it is often necessary to take memory into account. We briefly review instances of this, in order of increasing complexity:
   - Each cell stores a number of typical fixed constants to be used in its computations. In order to achieve correct reconfiguration, it is sufficient to add to each cell's store the constants proper to all cells that might reconfigure onto it, and to have error information activate correct constants as well as interconnection links.
   - Each cell has a limited number of registers whose contents are updated by simple computations. An extension of the previous instance may be considered, in which each cell has a number of register sets (one for each pair of logical indices it may assume) and it updates the contents of all registers at each computation step.
   - Each cell has a local storage facility larger than a register set. It becomes unthinkable to duplicate storage in each cell; it is necessary to foresee separate information paths for processing elements and for memories, and to make memories intrinsically robust by other, conventional techniques.
3. The most relevant extension to our fault model would concern interconnections. The problem of possible faults in interconnections is presently a subject of study.

The work presented can be seen as a first proposal, leading to wider adaptations and developments.

REFERENCES

1. Disparte, C.P. "A Design Approach for an Electronic Engine Controller Self-Checking Microprocessor." *Proceedings of the 7th Euromicro Symposium,* Paris, 1981, pp. 243–248.
2. Tsao, M.M., et al. "The Design of C-fast: A Single Chip Fault Tolerant Microprocessor." *Proceedings of the 12th FTCS,* Santa Monica, 1982, pp. 63–69.
3. Cenker, R.P. et al. "A Fault-Tolerant 64K Dynamic RAM." *Digest of the ISSCC,* 22 (1979), pp. 150–151.
4. Fitzgerald, B.F., and E. P. Thoma. "Circuit Implementation of Fusible Redundant Addresses on RAMs for Productivity Enhancement." *IBM Journal of Research & Development,* 24 (1980), pp. 291–298.
5. Kung, H.T. "Why Systolic Architectures?" *Computer,* 15, no. 1 (Jan. 1982), pp. 37–46.
6. Snyder, L. "Introduction to the Configurable, Highly Parallel Computer." *Computer,* 15, no. 1 (Jan. 1982), pp. 47–56.
7. Fairbairn, D.G. "VLSI: A New Frontier for System Designers." *Computer,* 15, no. 1 (Jan. 1982), pp. 87–96.
8. Mangir, R.M., and A. Avizienis. "Fault-Tolerant Design for VLSI: Effect of Interconnect Requirements on Yield Improvement of VLSI Design." *IEEE Transactions on Computing,* C-31 (1982), pp. 609–615.

Conflict-free memory allocation for associative data files

by SVETLANA P. KARTASHEV
University of Nebraska-Lincoln
Lincoln, Nebraska
and
STEVEN I. KARTASHEV
Dynamic Computer Architecture, Inc.
Lincoln, Nebraska

ABSTRACT

For associative processing and databases characterized by sequential memory search, it is convenient to store a sequence of data files in a circulating memory since it is inexpensive and suitable for implementing sequential memory search algorithms. In this paper we discuss various allocation algorithms that allow g circulating memories to serve T processors, where g and T are selectable by programmer.

All allocation schemes introduced in this paper are described by a Diophantine equation whose solution, x, shows the distance between any two processors that are not in conflict when they access the same circulating memory. The paper presents a technique for finding a maximal set of noninterfering processors and conflict-free allocation techniques for various structures of data files. These techniques achieve very high performance characteristics, since

1. They allow the entire memory space of a circulating memory either to be completely filled with data files or to be filled with minimal memory overhead created to exclude interference between any pair of noninterfering processors.
2. All the memory allocations developed are conflict-free.
3. During one memory revolution, the entire content of each circulating memory can be completely fetched by a set of noninterfering processors.

A. INTRODUCTION

For associative processing and databases characterized by sequential memory search, it is convenient to store a sequence of data files in a circulating memory, since it is inexpensive and suitable for implementing memory search algorithms. A circulating memory means a memory containing T^* words and possessing the following properties (Figure 1):

1. During each clock period the entire memory made of T^* words is shifted so that it can read out a new output word and write in a new input word.
2. To access the entire memory made of T^* words requires T^* clock periods.

The application of circulating memories for associative sorting and data searches stems from the fact that the circulation principle of this memory ideally conforms to the whole class of search algorithms that typically include the following steps:

1. Fetch the output word, w, in a word queue.
2. Compare w with the key word, k.
3. If $w \neq k$, go to step 1.
4. If $w = k$, end.

The modification of data files stored in a circulating memory can be performed by writing a new word to the current input word. In one revolution having T^* clock periods, the entire memory content can be modified.

Assume that in a circulating memory each data file is represented by a sequence of memory words being received by the buffer unit connected with the communication channel.

To take into account the various communication delays introduced by the communication network in data transmission, assume that the data words contained in the same data file are stored in sequence with a shifting distance from one to the next, $d \geq 1$, where the integer d is selectable by a programmer. A pair of adjacent data words from the same file may have a constant or variable d. (The case $d = 1$ means consecutive word storage.) Figure 3(a) shows a data file, $\{w_0, w_1, w_2, w_3\}$, made of 4 words with a consistent shifting distance, $d = 2$. In Figure 3(b) we have a 3-word file, $\{w_0, w_1, w_2\}$, characterized by a variable distance, d, between words, that is, $d(w_0, w_1) = 2$ and $d(w_0, w_2) = 5$, etc.

In this paper we discuss various allocation algorithms that allow g circulating memories to serve T processors, where the programmer may select $g = [\gcd(T, T^*)/j]$, where gcd is the greatest common divisor, by selecting T the number of processors, T^* the number of words in each memory, and j the allocation index related to the size of a data file that is fetched during each memory revolution. Also, as we have already said, the programmer may select a variable or constant distance d between a pair of adjacent words from the same file to characterize communication delay in a word broadcast via communication channel.

All allocation schemes introduced in this paper are described by a Diophantine equation* whose solution, x, shows the distance between any two processors that are not in conflict when they access the same circulating memory. The paper presents a technique for finding a maximal set of noninterfering processors and conflict-free allocation techniques for various structures of data files. These techniques achieve very high performance characteristics, since

1. They allow the entire memory space of a circulating memory either to be completely filled with data files or to be filled with minimal memory overhead created to exclude interference between any pair of noninterfering processors.
2. All the memory allocations developed are conflict-free.
3. During one memory revolution, the entire content of each circulating memory can be completely fetched by a set of noninterfering processors.

The relationship of this paper to other works on memory management is as follows:

1. (a) By creating noninterfering sets of processors that may access the same circulating memory in a conflict-free manner and (b) by forming various structures of data files for the noninterfering sets of processors, this paper contributes to the theory of shift-register sequences, since to form such processor sets and to find the data files structures for them requires application of interesting circular properties exhibited by shift-register sequences. This material connects this paper with earlier research.[2–7]

2. By developing conflict-free memory allocation techniques for a multiprocessor system that includes circulating memories, this paper is connected with the literature on conflict-free and parallel memory access.[8–12] Its contribution to the topic is in the development of conflict-free parallel allocation algorithms for a set of circulating memories connected to a multiprocessing system.

Section B outlines the problem of conflict-free memory allocation for a circulating memory and performs classification of data files and allocation schemes. Accordingly, data

*In general, by a Diophantine equation we mean a linear equation, $ax + by = c$, where a, b, and c are integers.[1]

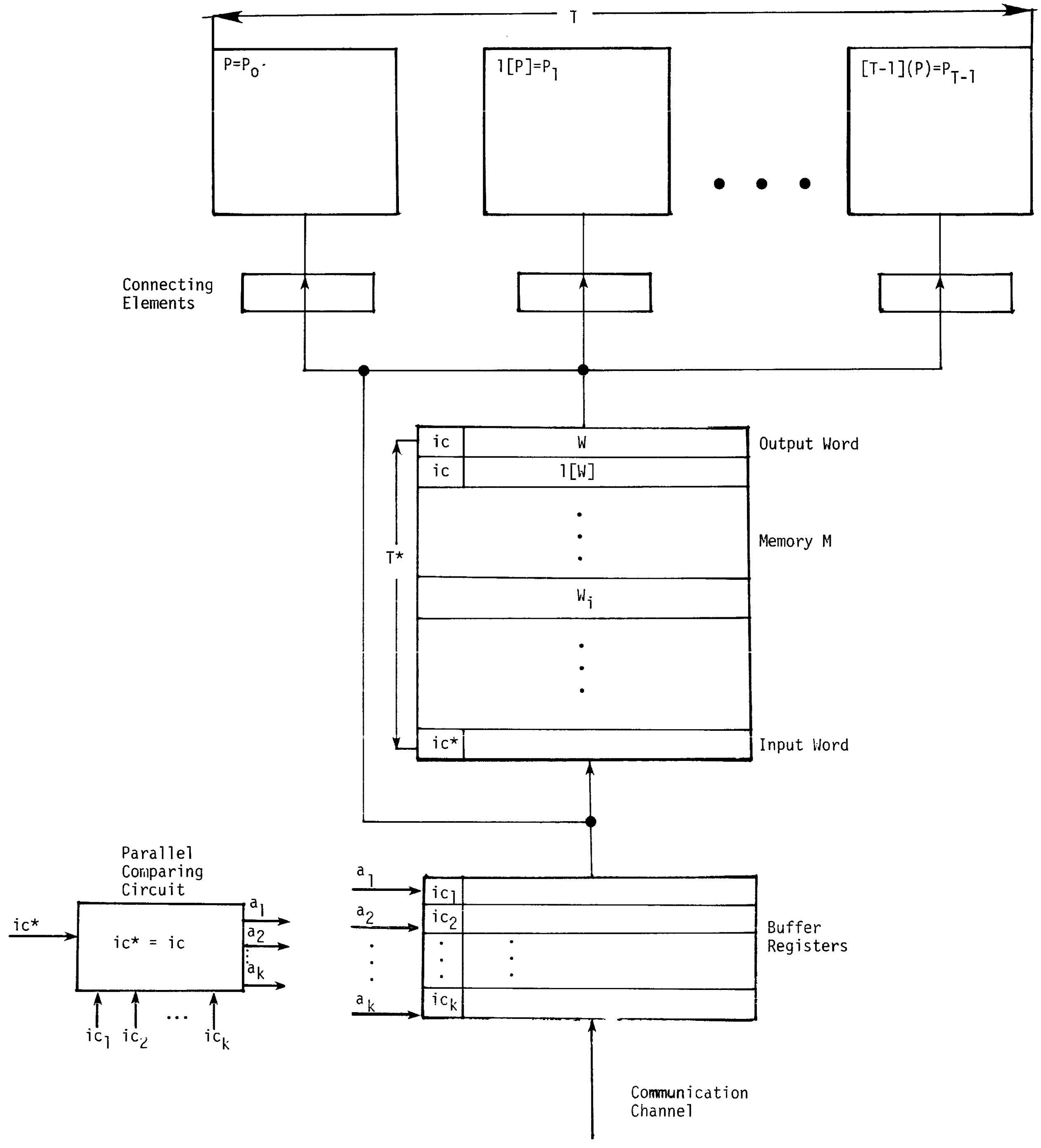


Figure 1—Block diagram

files can be minimal and nonminimal, where a minimal file includes only those memory words that are connected with the same processor during consecutive memory revolutions. To implement conflict-free memory access these words must be included in the same file, whose size is thus *minimal.* A *nonminimal file* properly contains several minimal files. The structure of nonminimal files is found in this paper. A nonminimal file can be *regular* if it is characterized by the same shifting distance, d, between its entries and *irregular* if shifting distance d is variable.

Section C introduces the structure of a minimal file. Section D presents a conflict-free file allocation for minimal data files. This allocation is very efficient since the entire memory space is filled with data words and the set of noninterfering processors contains a maximal number of members, $f = T^*/\gcd(T, T^*)$, where T is the number of processors, T^* the number of words in each memory.

Section E discusses the memory allocation for nonminimal data files. Section E.1 describes the structure and allocation procedures for regular nonminimal files, Section E.2 for irregular nonminimal files. Both allocations are conflict-free and achieve extremely high utilization of the available memory space. For regular files, no memory overhead is created. For irregular files, the number of unused memory words or memory overhead is minimal.

Section F discusses parallel operation of several circulating memories in the same multiprocessing system and makes conclusions about the importance of these allocations.

B. PROBLEM OF MEMORY ALLOCATION

In this section we will outline the problem of file allocation for circulating memories and classify the various data files that can be stored in them. In general, to take into account the various communication delays introduced by the communication channel, assume that each data file may take nonconsecutive words of a circulating memory.

Since a circulating memory may output one word per clock period, it is necessary to assume that during each clock period a word, w_j, issued by the circulating memory, M, may be received by a *new* processor, P_i, where P_{i-1} has received word w_{j-1} (Figure 1). (Here i changes mod-T, j changes mod-T^*, where T is the number of processors connected to the memory and T^* is the number of words or the *period of circulating memory*.) This assumption will lead to the minimization of the total time required to fetch T^* words by T processors, because this time will coincide with the time of one revolution of the memory. Indeed, an alternative organization whereby each processor, P_i, fetches its data file alone following the fetches completed by processor P_{i-1} (mod-T) will lead to a situation in which each data file will be fetched during one memory revolution, since each file consists of nonconsecutive memory words. Therefore, if the memory stores K job files, they will be fetched only after K revolutions of the memory.

However, to perform a minimization of the total time to access a queue made of K data files and stored in one circulating memory, the following problem has to be solved: Given T processors, $P_0, P_1, \ldots, P_{T-1}$ connected with a circulating memory M that stores the array of T^* data words (Figure 1). Generally, $T \neq T^*$. Each clock period, memory M may perform the following actions in parallel:

1. M may send its new output word via the output channel to a new processor, P_i (mod-T).
2. M may broadcast to the location of the input word either a current output word or a new input word received via the communication channel.

In this scheme buffer registers are registers that store incoming data words received from the communication channel. Each word is identified with the file identification code, ic, that is understood as the code of the data file containing this word.

Updating the input word of the memory, M, with a new data word is performed as follows. Since the circulating memory is already filled with data files, its current input word also stores some file identification code ic^*. This code is sent to a parallel comparing circuit that performs equality comparison of the current ic^* received by the memory input word with all the ic's of the data words stored in buffer registers. A buffer register for which $ic^* = ic$ transfers its word to the current input word. If several buffer registers store the same ic such that $ic^* = ic$, then the equality signal, α_i, generated as a result of $ic^* = ic$ transfers to the input word a data word belonging to the register with the smallest address. As for the remaining registers with the same ic, their contents are shifted, so that the register with the largest address is released and cleared in order to receive a new incoming message from the communication channel.

Assume that the circulating memory stores K data files, each of which may consist of nonconsecutive memory words. Develop such techniques of file storage in the memory that the following conditions are satisfied:

C1. *No file interference arises;* if two data files, DF_m and DF_l, are accessed by two processors, P_i and P_j, then $DF_m \cap DF_l = \phi$, that is, for each data file, every memory word contained in it is accessed by one processor only.

C2. *Minimal time access to all data files* is achieved, whereby a minimal number of dummy clock periods is created during memory revolutions when the processors fetch no data words.

C3. *Independence of allocation techniques of the memory revolutions* is achieved, whereby if a data file DF_i is accessed by processor P_k, then during consecutive memory revolutions the relative address of every word w contained in DF_i must remain unchanged in order to be accessed only by P_k. This processor, however, may fetch an updated word w if memory M updates its content at this address.

B.1 Noninterfering and Least-Interfering Allocations

If the entire memory space is allocated to data files in such a way that all memory words are included in noninterfering data files, then the allocation will be called *noninterfering*. For noninterfering allocation, no dummy clock periods are created, since the entire memory space is occupied with data words.

If, on the other hand, there are s common words in a pair of data files, they may be marked with a special bit ($C = 1$) to exclude their access by the two processors, so that the files become noninterfering. However, in accessing them the memory loses s clock periods during each revolution. This allocation will be called *s-interfering*. If s is the least possible integer, then the allocation becomes *least interfering*.

B.2 Minimal and Nonminimal Data Files

It should be noted that the use of the condition C3 in allocation leads to the appearance of a so-called *minimal data* *file, MF*(P), accessed by processor P and defined as follows: $MF(P)$ includes those memory words that are connected with P during consecutive memory revolutions. That is, if P fetches word w' during one memory revolution and word w'' during another revolution, then to make the allocation strategy independent of memory revolutions words w' and w'' must be included in minimal file handled by P.

Obviously, to implement C3, any nonminimal file, $NF(P)$, accessed by P must include $MF(P)$ as its nucleus; that is, $MF(P) \leq NF(P)$.

Therefore, allocation for two types of files should be considered:

1. The minimal file, $MF(P)$, since any other data file handled by P includes $MF(P)$
2. A nonminimal file, $NF(P)$, handled by P, because a correct selection of the NF size allows one to minimize the number of revolutions required to fetch any actual data file, $DF(P)$, that cannot be fetched during one memory revolution

B.3. Classification of Allocation Schemes

This paper considers two types of memory allocation: the minimal file allocation and the nonminimal one.

For the minimal file allocation, this paper finds that the size of a minimal file, MF, is determined as $f = T^*/\beta$, where $\beta = \gcd(T, T^*)$.

For the nonminimal file allocation, the paper finds the structure of each nonminimal file, NF. This must necessarily include a *primitive file, PF*, consisting of j words called the *allocation index*, selected by the programmer. The size of the nonminimal file is then determined as $j \cdot f = j \cdot T^*/\beta$. Thus if $j = 1$, the nonminimal allocation becomes minimal (Figure 2).

Each nonminimal file can be further classified as either *regular* or *irregular*. A regular file is specified by the same shifting distance d between any two consecutive memory words in its primitive file. An irregular file is specified by arbitrary shifting distances d_i between consecutive memory words, w_i and w_{i+1}, in the primitive file. For instance, in Figure 3(a) we have a regular primitive file, PF, that includes words w_0, w_1, w_2, and w_3, such that the shifting distance d between any two consecutive words is the same, $d = 2$. Thus, $PF = \{0, d, 2d, \ldots, (j-1)d\}$, if these distances are measured from the current output word w. In Figure 3(b), we have the irregular primitive file $PF = \{0, 2, 5\}$, since shifting distance between w_0 and w_1 is $d_1 = 2$ and between w_1 and w_2 is $d_2 = 3$. (For notational simplicity, all files will be represented by a set of shifting distances between a current output word w and all other memory words that are included in a file, where 0 is a shifting distance between w and w' if $w = w'$.)

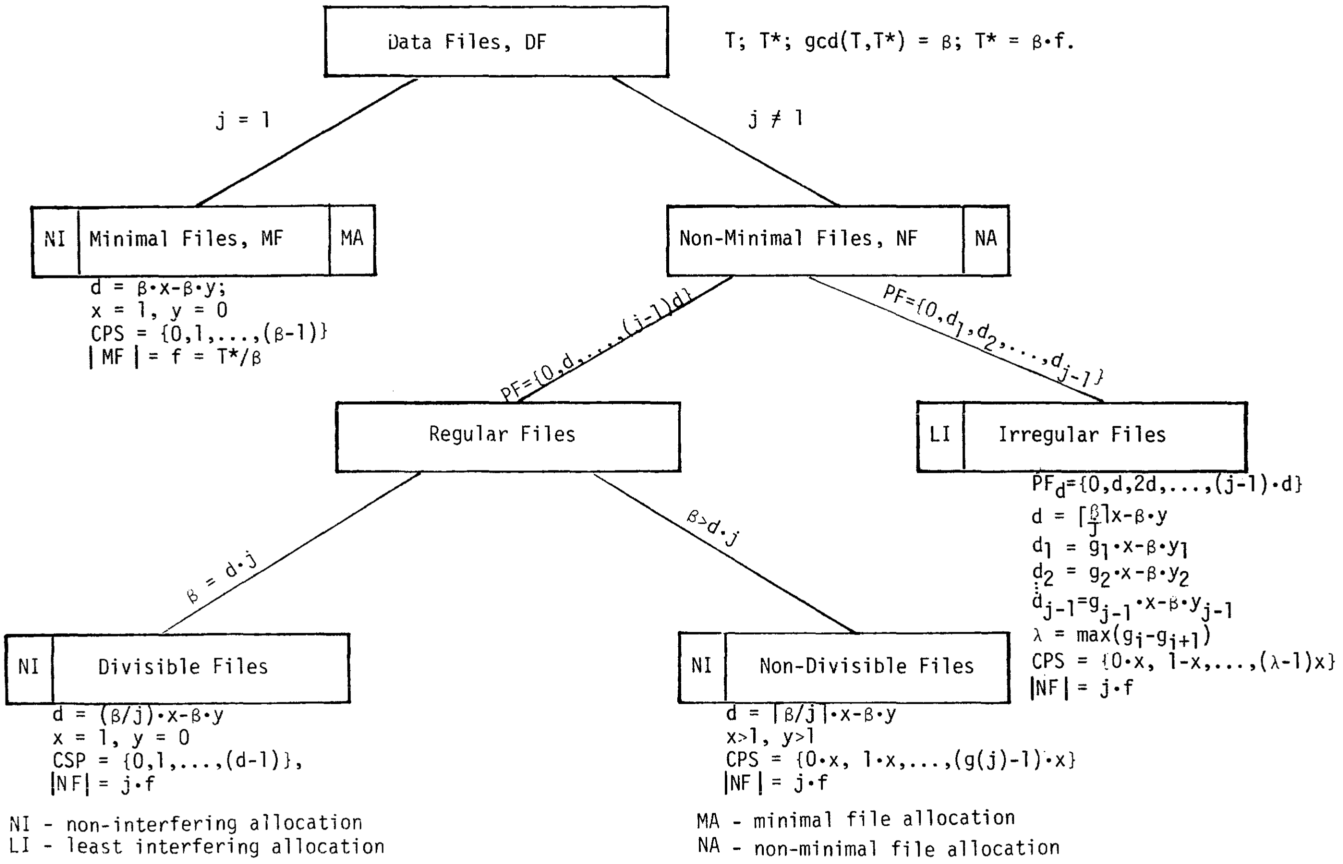


Figure 2—Classification of files stored in a circulating memory

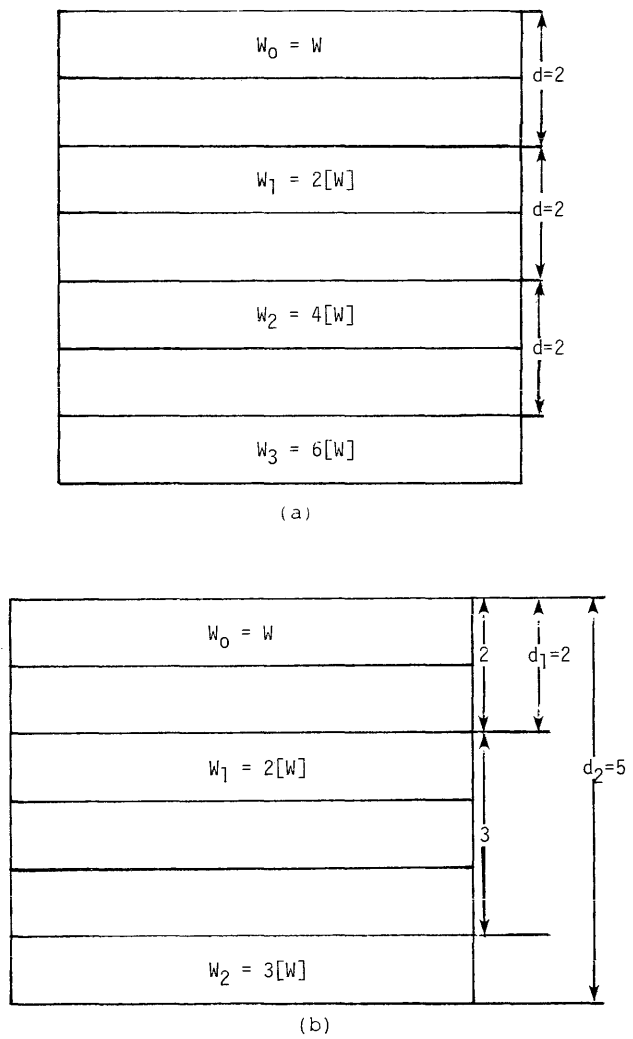


Figure 3—(a) Regular primitive file PF = [0, 2, 4, 6], (b) Irregular primitive file PF = [0, 2, 5]

Regular nonminimal files are further subdivided into divisible files, if $\beta = \gcd(T, T^*) = d \cdot j$, and nondivisible files, if $\beta = \gcd(T, T^*) > d \cdot j$, as shown in Figure 2.

This paper studies conflict-free allocations for all data files of the classification in Figure 2. The paper shows that for all files but the irregular ones the allocation creates no memory overhead if it is described by the Diophantine equation

$$d = g(j) \cdot x - \beta \cdot y,$$

where d is the permanent distance between two consecutive memory words belonging to the file of the same type (minimal and nonminimal regular); $g(j) = [\beta/j]$ is the number of processors that fetch noninterfering files—these processors form a noninterfering processor set, $NPS = \{P^{(0)}, p^{(1\cdot x)}, p^{(2x)}, \ldots, p^{(g(j)-1)\cdot x}\}$; x is the processor displacement that equals the shifting distance between two consecutive processors from the NPS set; and j is the allocation index, selected by the programmer. For minimal files, $j = 1$ and $d = \beta \cdot x - \beta \cdot y$. This implies that the only solution of the Diophantine equation is $x = 1$, $y = 0$. Thus the noninterfering processor set contains $g(1) = \beta$ processors with the displacement $x = 1$ from each other.

For the nonminimal divisible files $j \neq 1$ and $d = g(j) \cdot x - \beta \cdot y = (\beta/j) \cdot x - \beta \cdot y$. This equation has also only one solution, $x = 1$, $y = 0$, because β is divided by j and $d = \beta/j$ by definition. Thus the noninterfering processor set contains $d = \beta/j$ processors with the shifting distance $x = 1$ from each other.

For nonminimal nondivisible files, since $d \cdot j < \beta$, the Diophantine equation has a nontrivial solution, $x > 0$, $y > 0$, provided $\gcd([\beta/j], \beta) = 1$ and $\gcd(\beta, x) = 1$. Thus the noninterfering processor set NPS will have $g(j)$ members, where $g(j) = [\beta/j]$ and the shifting distance between any two consecutive processors from NPS is $x > 1$.

For irregular files, the allocation creates a minimal memory overhead, that is, it is the least interfering because it is described by the Diophantine equation that provides the best approximation to the closest regular allocation that is noninterfering.

The paper shows that all allocation techniques are completely deterministic and described by the strict algorithmic procedures that it presents.

B.4 Parallel Operation of Several Circulating Memories

Since a noninterfering processor set, NPS, that allows obtaining noninterfering data files includes only $g(j)$ processors, the remaining $T - g(j)$ processors do not participate in file fetches from the same circulating memory, M. This paper describes a procedure to prevent their idleness by connecting these processors with other circulating memories, so that all processors will be fetching data files and all the memories will be provided with either noninterfering or the least interfering allocations.

As a result, the allocation techniques presented in this paper achieve very high performance characteristics since

1. They allow obtaining either noninterfering or least-interfering memory allocation, in which the entire memory space of each memory, M, is either completely filled with noninterfering job files accessed by $g(j)$ processors or a minimal number of memory words is not used.
2. T processors can be partitioned into $T/g(j)$ groups, each of which is served by one circulating memory.

C. STRUCTURE OF A MINIMAL FILE AND FILE INTERFERENCE PROBLEM

In this section we will find the size of a minimal file, MF, and outline the problem of file interference.

Let processor P fetch word w. Since the period T^* of M is not the same as the number of processors, T, during each revolution of M, one processor, P, will fetch a new word of M. We find the number of such words and their relative locations with respect to the original word w. Obviously all these words should be included in the minimal file of P.

Another problem that will be solved is finding out what processors fetch the same word w during consecutive revolutions of M.

C.1 Structure of the Minimal File

As was indicated in Section B, the following rule of accessing will be used: If processor P_i accesses word w_k, then the next word, w_{k+1}, will be accessed by P_{i+1}, where i changes mod-T and k changes mod-T^*.

Therefore during word fetches from memory M, processors are forming *processor sequence PS* of period T, and memory words are forming memory sequence MS^* of period T^*:

$$PS = \{P, 1[P], \ldots, (T-1)[P]\}$$

$$MS^* = \{w, 1[w], \ldots, (T^*-1)[w]\}.$$

Here the sequentiality of different P's and w's in PS and MS, respectively, is understood as follows: If processor P fetches word w, then the next processor, $1[P]$, fetches word $1[w]$. Iterative application of this rule leads to processor $i[P]$ accessing word $i[w]$, where i shows the displacement or the shifting distance between P and $i[P]$ or w and $i[w]$, respectively.

Obviously, for processors $T[P] = P$; that is, after T shifts the same processor will fetch word $T[w]$. Likewise, for memories $T^*[w] = w$, that is, after T^* shifts the same memory location, w, is accessed by processor $T^*[P]$.

For convenience of notation, we denote the event of P fetching word w as $P \rightarrow w$, where P is called *source, w* is called *destination.*

Let us find the least common multiple of T and T^*, $\mathrm{lcm}(T, T^*)$, $f = \mathrm{lcm}(T, T^*)/T$ and $f^* = \mathrm{lcm}(T, T^*)/T^*$. The significance of the numbers f and f^* is that the f integer shows the minimal number of different words that are connected with processor P during consecutive revolutions of memory M.

These words are found as follows. If $P \rightarrow w$, then

$$T[P] \rightarrow T[w],$$

$$2T[P] \rightarrow 2T[w],$$

$$\vdots$$

$$(f-1)[P] \rightarrow (f-1)[w]$$

For these fetches, all sources are the same since T is the period of the processor sequence. Thus $P = T[P] = 2T[P] = \ldots = (f-1)[P]$. The destinations, however, are not the same, since in general T^*, the period of memory sequence does not divide T, that is, $T^* \not| \, T$.

Thus the minimal file of P, $MF(P)$, contains the words

$$MF(P) = \{w, T[w], 2T[w], \ldots, (f-1)\cdot T[w]\}. \quad (1)$$

From the constructing procedures used in finding $MF(P)$, this is a minimal set, since in its construction the only assumption that was made was that a processor P fetches only one word, w, from this file. All other members of $MF(P)$ have been found by iterative application of the $T[w]$ rule.

C.2 Fetches of Minimal File During One Memory Cycle

In order to provide that all words from the minimal file of P be fetched during one memory cycle, it is necessary to find values for their relative positions, $T, 2T, \ldots, (f-1)T$, that do not exceed T^*; that is, it is necessary to reduce them by mod T^*. This can be easily done using the following well-known properties of congruences: "When each member of a complete set of residues $\{0, 1, 2, \ldots, p-1\}$ is multiplied by an integer k which is not a multiple of p to yield $\{0, k, 2k, \ldots, (p-1)\cdot k\}$ and when these numbers are reduced mod p, the resulting remainders are precisely the numbers in the set $\{0, 1, 2, \ldots, p-1\}$ except possible to the order."[1]

In our case, we reduce all numbers, $\{0, T, 2T, \ldots, (f-1)\cdot T\}$ by mod-T^*. Since $f\cdot T = f^*\cdot T^* = \mathrm{lcm}(T, T^*)$, $T = f^*\cdot\beta$, $T^* = f\cdot\beta$, where $\beta = \gcd(T, T^*)$. All members of the minimal file are then reduced by mod-$T^* = f\cdot\beta$ and not by mod-f as in the property of congruences formulated for $\beta = 1$. Since for the general case $\beta \geq 1$, the result of mod-T^* reduction will be an integer including two factors, β and i. Thus the minimal file of P, $MF(P)$, will be given as:

$$MF(P) = \{w, \beta[w], 2\cdot\beta[w], \ldots, (f-1)\cdot\beta[w]\}, \quad (2)$$

where $\beta = \gcd(t, T^*)$.

D. NONINTERFERING FILE ALLOCATION FOR MINIMAL PROCESSOR FILES

In this section we will discuss the procedures for noninterfering file allocation of minimal processor files $MF(P)$, $MF(P) = \{0, \beta, 2\beta, \ldots, (f-1)\cdot\beta\}$. Let us find the set of *noninterfering processors,* that is, those of which any pair of minimal files do not interfere, or have empty intersections. This can be done by applying shift iterations to the set $MF(P)$ as follows: 1-bit shift, $1[MF]$, of MF is the minimal file accessed by the processor $P^{(1)} = 1[P]$. This file has the form $MF(P^{(1)}) = 1[MF] = \{1, \beta+1, 2\beta+1, \ldots, (f-1)\beta+1\}$.

Similarly one can obtain 2-bit shift, $2[MF]$, of MF, accessed by the processor $P^{(2)} = 2[P]$, etc. The number of noninterfering processors is β. Their locations relative to the P processor are

$$P, 1[P], 2[P], \ldots, (\beta-1)[P].$$

These locations form a noninterfering processor set,

$$NPS = \{P, 1[P], 2[P], \ldots, (\beta-1)[P]\}.$$

Thus there exists a noninterfering file allocation if the memory M is accessed by precisely β processors forming a noninterfering processor set,

$$NPS = \{P, 1[P], \ldots, (\beta - 1)[P]\}.$$

Example: Let us construct the noninterfering processor set, *NPS*, for $T = 15$, $T^* = 35$, $\beta = \gcd(15, 35) = 5$, $f = T^*/\beta = 7$. We obtain the noninterfering processor set, $NPS = \{P, 1[P], 2[P], 3[P], 4[P]\}$, where these processors are allocated with the following minimal data files. Since $f = 7$, $f - 1 = 6$:

$$\begin{aligned}
MF(P) &= \{0, 5, 2\cdot 5, 3\cdot 5, 4\cdot 5, 5\cdot 5, 6\cdot 5\}\\
&= \{0, 5, 10, 15, 20, 25, 30\},\\
MF(P^{(1)}) = MF(1[P]) &= \{1, 6, 11, 16, 21, 26, 31\},\\
MF(P^{(2)}) = MF(2[P]) &= \{2, 7, 12, 17, 22, 27, 32\},\\
MF(P^{(3)}) = MF(3[P]) &= \{3, 8, 13, 18, 23, 28, 33\},\\
MF(P^{(4)}) = MF(4[P]) &= \{4, 9, 14, 19, 24, 29, 34\}.
\end{aligned}$$

Therefore, there is no interference between minimal data files accessed by the processors from the noninterfering processor set. Also memory M is completely filled with information. Thus, the allocation is noninterfering.

E. NONMINIMAL FILES

In this section we will discuss a noninterfering file allocation for nonminimal data files.

Each nonminimal file is specified as follows: for each word w fetched by P, there exists a minimal processor file, $MF_w(P) = \{w, \beta[w], 2\beta[w], \ldots, (f-1)\beta[w]\}$; therefore, to give a nonminimal file $NF(P)$ is

1. To give only *j primitive words,* that is, those words any pair of which are not in the same $MF(P)$,
2. To construct the minimal data file for each primitive word.

The total number of words in $NF(P)$ is $j \cdot f$. It then follows that each nonminimal job file accessed by P is specified by primitive file $PF(P)$ made of j primitive words; $PF(P) = \{w, d_1[w], d_2[w], \ldots, d_{j-1}[w]\}$, where d_i is the shifting distance between a current output word w and another primitive word. Obviously, it is possible to select primitive words such that each $d_i < \beta$, inasmuch as the distance between each primitive word and its derivatives in the minimal job file originated by it is a multiple of $\beta(1\cdot\beta, 2\cdot\beta, \ldots, (f-1)\beta)$. For convenience, all d's are ordered as $d_i < d_{i+1}$ $(i = 1, \ldots, j-1)$. Therefore, the nonminimal job file, $NF(P)$, can be expressed in the following form: Given words w, $d_1[w]$, $d_2[w], \ldots, d_{j-1}[w]$, that must be accessed by P, then all the words accessed by P during consecutive memory revolutions are given as shown in Figure 4. The procedure of obtaining *NF* from *PF* is called β-expansion where $\beta = \gcd(T, T^*)$.

Example: Let $T = 30$ processors be connected with the circulating memory containing $T^* = 105$ words. Let $\gcd(T, T^*) = \gcd(30, 105) = 15$ and $f = 105/15 = 7$. Suppose that a

$$NF(P) = \left|\begin{array}{l}
0,\ \beta,\ 2\beta,\ \ldots, (f-1)\beta\\
d_1,\ d_1+\beta, d_1+2\beta, \ldots, d_1+(f-1)\beta\\
d_2,\ d_2+\beta, d_2+2\beta, \ldots, d_2+(f-1)\beta\\
.\\
.\\
.\\
d_{j-1}, d_j+\beta, d_j+2\beta, \ldots, d_j+(f-1)\beta
\end{array}\right|$$

Figure 4—Words accessed by P during consecutive memory revolutions

processor P must fetch the following primitive words: $PF(P) = \{w, d_1[w], d_2[w]\}$, where $d_1 = 2$ and $d_2 = 5$. Thus during each revolution it fetches a nonminimal job file, $NF(P)$, that includes the following words given by their positions relative to w:

$$NF(P) = \left|\begin{array}{l}
0,\ 15,\ 2\cdot 15,\ 3\cdot 15,\ 4\cdot 15,\ 5\cdot 15,\ 6\cdot 15\\
2,\ 2+15,\ 2+2\cdot 15, 2+3\cdot 15, 2+4\cdot 15, 2+5\cdot 15, 2+6\cdot 15\\
5,\ 5+15,\ 5+2\cdot 15, 5+3\cdot 15, 5+4\cdot 15, 5+5\cdot 15, 5+6\cdot 15
\end{array}\right|$$

This file will contain $3\cdot 7 = 21$ words. Thus for a noninterfering file allocation, the memory can be accessed by not more than five processors $(5 = 105/21)$. Since this allocation is irregular, however, it is the least interfering allocation and the noninterfering processor set will contain six processors separated by displacement $x = 8$.

In Sections E.1 and E.2 we will study noninterfering and least interfering file allocations for nonminimal job files, regular and irregular. Before developing some results for the general case $j > 1$ and $d_1 \nmid d_i$ $(i = 2, \ldots, j-1)$, that is, d_1 does not divide any d_i, let us consider a particular case of regular files whereby the primitive file $PF(P)$ is given as

$$PF(P) = \{0, d, 2d, \ldots, (j-1)d\},$$

where $j \cdot d \le \beta = \gcd(T, T^*)$. This file is called a *regular nonminimal file* since its primitive file is made of all equally distant primitive words, that is, $PF = \{0, d, 2d, \ldots, (j-1)d\}$.

E.1 Regular Nonminimal Files

Using a β-expansion procedure whereby each primitive word generates its minimal file, one obtains the regular nonminimal file

$$NF = \left|\begin{array}{lllll}
0, & \beta, & 2\beta, & \ldots, & \beta(f-1)\\
d, & d+\beta, & d+2\beta, & \ldots, & d+\beta(f-1)\\
2d, & 2d+\beta, & 2d+2\beta, & \ldots, & 2d+\beta(f-1)\\
. & & & & .\\
.\\
.\\
\multicolumn{5}{l}{(j-1)d, (j-1)d+\beta, (j-1)d+2\beta, \ldots, (j-1)d}\\
\multicolumn{5}{r}{+\beta(f-1)}
\end{array}\right|$$

As was shown in Figure 2, one can have two types of regular nonminimal files:

1. *Divisible regular files,* in which $j \cdot d = \beta$, where $\beta = \gcd(T, T^*)$, and
2. *Nondivisible regular files,* in which $j \cdot d < \beta$.

The next two sections will discuss noninterfering allocation for these types of files.

E.1.1. Divisible regular files. For divisible regular files there exists a noninterfering file allocation. A noninterfering processor set consists of d processors given by the following relative distances to each other: $NPS = \{0, 1, 2, \ldots, d-1\}$.

Example: Let us construct divisible regular files for the following processor-memory interconnection scheme: $T = 27$; $T^* = 45$; $j = 3$. This means that since $\beta = \gcd(T, T^*) = \gcd(27, 45) = 9$, $f = T^*/\beta = 45/9 = 5$, and $d = \beta/j = 9/3 = 3$, one can select a noninterfering nonminimal file allocation. Each nonminimal file, NF, contains $|NF|$ members, where $|NF| = f \cdot j = 5 \cdot 3 = 15$. The set of noninterfering processors is given as $NPS = \{0, 1, 2\} = \{P, 1[P], 2[P]\}$.

This allocation is noninterfering since it introduces no dummy clock periods in accessing and achieves a complete filling of the available memory space. These nonminimal files are given in Table I.

TABLE I—Noninterfering allocation of nonminimal, divisible irregular files

		$\beta l^{+} + s^{**}$					
$NF^{(s)}$	$d{\cdot}m^*$	$s=0$	$0{\cdot}\beta$	$1{\cdot}\beta$	$2{\cdot}\beta$	$3{\cdot}\beta$	$4{\cdot}\beta$
	0		0	9	18	27	36
$NF^{(0)}$	d		3	12	21	30	39
	$2d$		6	15	24	33	42
		$s=1$	$0{\cdot}\beta+1$	$1{\cdot}\beta+1$	$2{\cdot}\beta+1$	$3{\cdot}\beta+1$	$4{\cdot}\beta+1$
	0		1	10	19	28	37
$NF^{(1)}$	d		4	13	22	31	40
	$2d$		7	16	25	34	43
		$s=2$	$0{\cdot}\beta+2$	$1{\cdot}\beta+2$	$2{\cdot}\beta+2$	$3{\cdot}\beta+2$	$4{\cdot}\beta+2$
	0		2	11	20	29	38
$NF^{(2)}$	d		5	14	23	32	41
	$2d$		8	17	26	35	44

+$m = 0, \ldots, j-1$
**$l = 0, \ldots, f-1$

E.1.2 Nondivisible Regular Files. By a nondivisible regular job file, NF, we mean a nonminimal file accessed by processor P and specified with the following parameters:

1. Its primitive file is $PF = \{0, d, 2d, \ldots, (j-1)d\}$, where d is the equal shifting distance parameter.
2. $d \cdot j < \beta$, where $\beta = \gcd(T, T^*)$, T is the number of processors, and T^* is the number of words of circulating memory M.

Since $d \cdot j < \beta$, j may not be a divisor of β. Thus the distance d will be presented as

$$d = \lceil \beta/j \rceil \cdot x - \beta \cdot y .$$

We obtained a linear Diophantine equation called the *modular equation.* Applying a well-known theorem of number theory to a solution of the Diophantine equation,[1] we found that this modular equation has a solution if and only if $\gcd(\lceil \beta/j \rceil, \beta)$ is a divisor of the fixed distance d. Therefore, if β, j, and d are given, not all d's can have x and y solutions of the modular equation. However, if for β, j, and d there is an integer solution, then there exists a noninterfering file allocation whereby no dummy clock periods are introduced in file fetches.

Theorem 1. If there exists an integer solution x and y of the modular equation $d = g(j) \cdot x - \beta \cdot y$, where $g(j) = \lceil \beta/j \rceil$, and $\gcd(\beta, x) = 1$, then there exists a noninterfering file allocation whereby a noninterfering processor set is $NPS = \{P^{(0)}, P^{(x)}, P^{(2x)}, \ldots, P^{((g-1)x)}\}$, where $P^{(ix)} = ix[P^{(0)}]$, and x is the root of the modular equation. Each processor, $P^{(ix)}$, but the last one, $P^{((g-1)x)}$, accesses a standard nonminimal file, $NF^{(ix)}$, containing $j \cdot f$ members where j is the index of allocation. In other words, if $P^{(0)}$ accesses $NF^{(0)}$ then any $P^{(ix)}$ but $P^{((g-1)x)}$ accesses $NF^{(ix)} = ix[NF^{(0)}]$. The last processor, $P^{((g-1)x)}$, fetches a smaller nonminimal file $NF_L{}^{(g-1)x}$ obtained from standard file $NF^{(g-1)x}$ by erasing the last L words, $L = g \cdot f \cdot j - T^*$, since these words are included in $NF^{(0)}$.

Corollary. For the modular equation $d = g(j) \cdot x - \beta \cdot y$ having $g(j) = \beta/j$, there exists the trivial solution $x = 1, y = 0$. Therefore, for minimal ($j = 1$) and nonminimal divisible files ($d = g(j) = \beta/j$), the noninterfering processor set includes $g(j)$ members with displacement $x = 1$; $NPS = \{P, 1[P], 2[P], \ldots, (g(j)-1)[P]\}$.

Example: Let memory M have $T^* = 42$ words. Suppose that this memory is attached to $T = 28$ processors. Suppose that $d = 3$ and $j = 3$, that is, for each nonminimal file NF its primitive file contains $j = 3$ primitive words with the same relative distance $d = 3$ from each other; $PF = \{0, d, 2d, \ldots, (j-1) \cdot d\} = \{0, d, 2d\} = \{0, 3, 6\}$. Since $\beta = \gcd(T, T^*) = \gcd(28, 42) = 14$ and $d \cdot j = 3 \cdot 3 = 9$, we have $\beta > d \cdot j$. Let us find $f = T^*/\beta = 42/14 = 3$. According to Theorem 1, there exists a noninterfering file allocation for which $L = 3$, that is, $L = g \cdot f \cdot j - T^* = \lceil \beta/j \rceil \cdot f \cdot j - T = 5 \cdot 3 \cdot 3 - 42 = 3$. Thus only three members of two nonminimal file sets are common. Let us find x and y roots; since $\gcd(\lceil \beta/j \rceil, \beta) = \gcd(\lceil 14/3 \rceil, 14) = \gcd(5, 14) = 1$, there exists an x and y solution of the modular equation $d = \lceil \beta/j \rceil \cdot x - \beta \cdot y = 5x - 14y = 5 \cdot 9 - 14 \cdot 3 = 45 - 42$. Therefore, processor displacement $x = 9$, and the noninterfering processor set is given as $NPS = \{0, x, 2x, \ldots, (\lceil \beta/j \rceil - 1)x\} = \{0, x, 2x, 3x, 4x\}$. Each processor of the NPS accesses the nonminimal file given in Table II and in Figure 5. (In Figure 5 each memory word is marked by the unique processor that fetches it.)

The only common words are words 0, 14, and 28 in files $NF^{(0)}$ and $NF^{(4x)}$. Assume that 0, 14, and 28 belong only to $NF^{(0)}$; for this reason, they are marked with a special bit E in Figure 5. $NF_L{}^{(4x)}$ is then obtained from $NF^{(4x)}$ by erasing words 0, 14, and 28 from it. Thus file allocation is noninterfering, since no memory words are common.

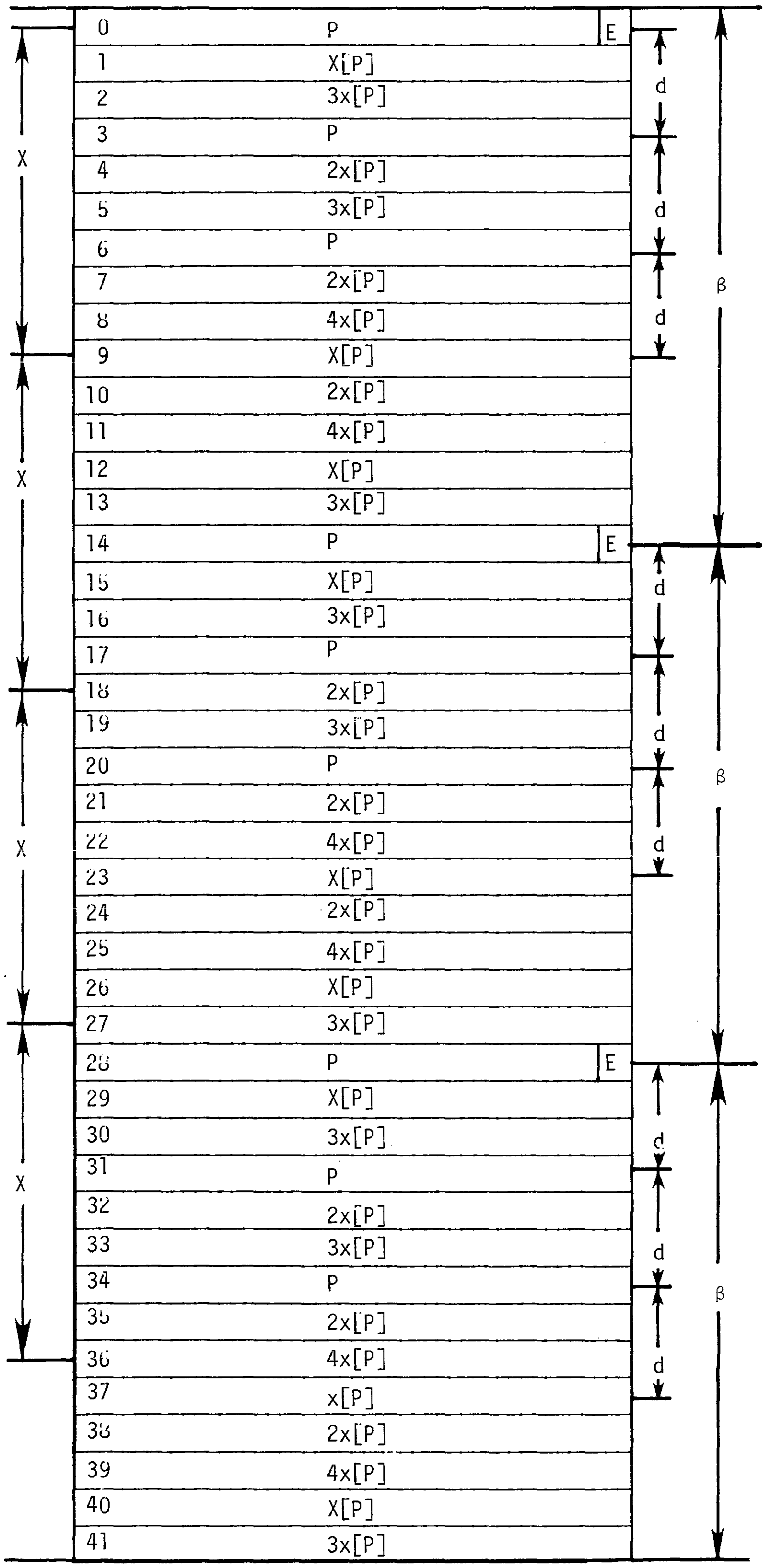


Figure 5—Noninterfering file allocation for nondivisible regular files $NF^{(0)}$, $NF^{(1 \cdot x)}$, $NF^{(2 \cdot x)}$, $NF^{(3 \cdot x)}$, $NF^{(4 \cdot x)}$ accessed by processors P, $x[P]$, $2x[P]$, $3x[P]$, and $4x[P]$, respectively

TABLE II—Noninterfering allocation of nonminimal, nondivisible regular files

$NF^{(sx)}$	$d \cdot m^*$	$\beta \cdot l^+ + s^* \cdot x$			
		$s=0$	$0\cdot\beta$	$1\cdot\beta$	$2\cdot\beta$
	0		0	14	28
$NF^{(0)}$	d		3	17	31
	$2d$		6	20	34
		$s=1$	$0\cdot\beta+x$	$1\cdot\beta+x$	$2\cdot\beta+x$
	0		9	23	37
$NF^{(x)}$	d		12	26	40
	$2d$		15	29	1
		$s=2$	$0\cdot\beta+2x$	$1\cdot\beta+2x$	$2\cdot\beta+2x$
	0		18	32	4
$NF^{(2x)}$	d		21	35	7
	$2d$		24	38	10
		$s=3$	$0\cdot\beta+3x$	$1\cdot\beta+3x$	$2\cdot\beta+3x$
	0		27	41	13
$NF^{(3x)}$	d		30	2	16
	$2d$		33	5	19
		$s=4$	$0\cdot\beta+4x$	$1\cdot\beta+4x$	$2\cdot\beta+4x$
	0		36	8	22
$NF^{(4x)}$	d		39	11	25
	$2d$		0	14	28

E.1.3 Generation of processor addresses for nonminimal file allocation. Consider the procedures for finding the address of a processor that has to fetch a current output word w issued by the memory. Since for the minimal and divisible regular files, the noninterfering processor set is given as $NPS = \{0, 1, 2, \ldots, (d-1)\}$, each dth word of the memory goes to the same processor. Thus generation of the processor addresses may be obtained with a mod-d counter that shows the address of a processor that has to receive a current output word.

For the nondivisible regular files, the procedure of finding a processor address becomes more complicated since two adjacent words, w and $1[w]$, should be fetched by the two processors that may not be adjacent in the NPS. For instance, in the example above, if word 3 is in $NF^{(0)}$ and is accessed by $P^{(0)}$ then the next word 4 should be accessed by $P^{(2x)} = P^{(18)}$ since it belongs to $NF^{(2x)}$ where $2 \cdot x = 18$ (Figure 5).

Therefore one must devise procedures for generating processor addresses for nondivisible regular files, because only then can one take advantage of elegant and attractive attributes of the allocation that are developed. We have devised these procedures, but will not be discussing them in this paper.

E.2 Irregular Files

A file is called irregular, if (a) it is a nonminimal and (b) its primitive file, PF, is given as $PF = \{0, d_1, d_2, \ldots, d_{j-1}\}$, where shifting distances $d_1, d_2, \ldots, d_j$ are not multiples of an integer d.

In this section we will study a file allocation of irregular files. It follows from Theorem 1 that generally there exists no noninterfering file allocation of irregular files. Indeed, by definition the modular equation $d = \lceil \beta/j \rceil x - \beta \cdot y$ has no integer solutions, x and y, because $d_1 \neq d$ and $d_2 \neq 2d, \ldots, d_{j-1} \neq (j-1)d$. Thus, no d can be found; this is not to mention other conditions such as $\gcd(\lceil \beta/j \rceil, \beta) = 1$ and $\gcd(x, \beta) = 1$.

However, if we find a close approximation of $PF = \{0, d_1, d_2, \ldots, d_{j-1}\}$ with some set $PF_d = \{0, d, 2d, \ldots, (j-1)d\}$ of equal distances it is possible to find a least-interfering file allocation. This means that it is possible to construct such a noninterfering processor set NPS that (a) each $P \in NPS$ fetches a nonminimal file, NF, given in Figure 4, and (b) between different files fetched by members of NPS there exists a minimal interference, that is, the minimal number of common words. Thereafter, each common memory word is marked with a special bit C to prevent its fetch by a processor.

For this reason, a dummy clock will be introduced into a file access each time a memory word with bit C is the output word, since no processor will fetch this word. The number of such dummy clocks will be minimal, since a given irregular allocation is represented with the best regular approximation that allows a noninterfering allocation according to Theorem 1.

As will be shown in Section E.2.2, the procedures for obtaining a noninterfering processor set NPS are deterministic; they are described with the use of an irregular allocation algorithm that ends with marking common words belonging to more than one job file with bit $C = 1$.

But before presenting an irregular file allocation algorithm we will introduce an example of irregular file allocation.

E.2.1 Example of irregular file allocation. Let memory M contain 105 words ($T^* = 105$) and be connected with $T = 30$ processors. Suppose that processor $P^{(0)}$ fetches the following primitive file: $PF^{(0)} = \{0, 2, 5\}$, so that $d_1 = 2$, $d_2 = 5$, and $j = 3$. Since d_1 does not divide d_2, $PF^{(0)}$ is irregular. The resulting nonminimal file, $NF^{(0)}$, will contain $j \cdot f = 21$ words, $j = 3$, $f = T^*/\gcd(T, T^*) = 105/15 = 7$. Using the equation in Figure 4, we construct the file in Table III.

If we select a noninterfering processor set as $NPS = \{0, x, 2x, 3x, 4x, 5x\}$, where $x = 8$, we will have a file allocation with the least interference. (A formal procedure for finding this x will be given in the next example.) Indeed, if processor $P^{(0)}$ accesses $NF^{(0)}$ given above, then processor $P^{(x)} = x[P]$ will access following $NF^{(x)}$ in Table IV. The processors $P^{(2x)}$, $P^{(3x)}$, $P^{(4x)}$, and $P^{(5x)}$ will access the $NF^{(2x)}$, $NF^{(3x)}$, $NF^{(4x)}$, and $NF^{(5x)}$ also in Table IV. As seen, the allocation is 21-interfering. Indeed, $NF^{(0)}$, $NF^{(x)}$, $NF^{(2x)}$, and $NF^{(3x)}$, have no common words at all; as for $NF^{(4x)}$, it has a common row of 7 words with $NF^{(0)}$; $NF^{(5x)}$ has a common row of 7 words with $NF^{(0)}$ (7 words), and a common row of 7 words with $NF^{(x)}$ (7 words). The common rows of $NF^{(4x)}$ and $NF^{(5x)}$ are squared. In all 21 words are common.

The minimal interference of this allocation is established as follows. If for the same memory containing 105 words a noninterfering allocation exists, it requires $105/21 = 5$ processors,

TABLE III—Noninterfering allocation of nonmimimal irregular files

	d	0	$1{\cdot}\beta$	$2{\cdot}\beta$	$3{\cdot}\beta$	$4{\cdot}\beta$	$5{\cdot}\beta$	$6{\cdot}\beta$
	0	0	15	30	45	60	75	90
NF^0	d_1	2	17	32	47	62	77	92
	d_2	5	20	35	50	65	80	95

each processor fetching $j \cdot f = 3 \cdot 7 = 21$ memory words. However, in accordance with Theorem 1, no noninterfering allocation exists. Our allocation requires six processors instead of five, that is, only one processor more than is required by a noninterfering allocation. For this allocation each processor fetches 21 words, so six processors will fetch $21 \cdot 6 = 126$ words. Since the memory stores 105 words, $126 - 105 = 21$ words are common. The number of common words is minimal, since the memory is accessed only by one processor more than is required by the noninterfering processor set that can be constructed for the noninterfering allocation.

E.2.2 Irregular file allocation algorithm. Given: Irregular primitive file $PF = \{0, d_1, d_2, \ldots, d_{j-1}\}$.

Find: Noninterfering processor set NPS, such that the resulting allocation includes all memory words and introduces the least interference between various nonminimal files accessed by individual processors from the set NPS.

Step 1. For a given irregular primitive file, $PF = \{0, d_1, d_2, \ldots, d_{j-1}\}$, find a set, $PF_d = \{0, d, 2d, \ldots, (j-1)d\}$, of equal distances such that intersection $PF \cap PF_d$ contains the maximal number of members. The set PF_d is called the *maximal approximation of PF.*

Step 2. Find $\lceil \beta/j \rceil = g(j)$, where $\beta = \gcd(T, T^*)$; T is the number of processors; T^* is the number of memory words; and j is the number of words in PF_d.

Step 3. Construct modular equation, $d = \lceil \beta/j \rceil x - \beta y = gx - \beta y$. Find its solution, x and y, such that $\gcd(\beta, x) = 1$ and $\gcd(g, \beta) = 1$. If no integer solution, x, y, can be found, then either change j in PF_d or select another $PF_{d'}$, and so on, until such d' and/or j' are found that $d' = g' \cdot x' - \beta \cdot y'$ has integer solution x' and y'.

Step 4. For each shifting distance, d_i, that is a member of irregular primitive file PF form the following equations: $d_1 = g_1 \cdot x - \beta \cdot y_1$; $d_2 = g_2 \cdot x - \beta \cdot y_2$; $\ldots$; $d_{j-1} = g_{j-1} \cdot x - \beta \cdot y_{j-1}$, where x is the fixed root of the modular equation $d = g \cdot x - \beta \cdot y$ found in Step 3. All these equations have solutions g_i, y_i $(i = 1, \ldots, j-1)$, respectively, since $d_i < \beta$ and $\gcd(x, \beta) = 1$, that is, $\gcd(x, \beta) | d_i$ $(i = 1, \ldots, j-1)$.

Step 5. Order numbers $g_1, \ldots, g_{j-1}$, obtained in Step 4 as $g'_1 > g'_2 > \ldots > g'_{j-1}$. The noninterfering processor set, NPS, will contain λ members where $\lambda = \max(g_i' - g_{i+1}')$ and g_i' and g_{i+1}' are two consecutive members of ordering $g_1' > g_2' > \ldots > g_{j-1}'$. And $NPS = \{P^{(0)}, P^{(x)}, P^{(2x)}, \ldots,$

TABLE IV—Noninterfering allocation of nonminimal irregular files

		$\beta{\cdot}l + y$						
	$y = x$	$0{\cdot}\beta + x$	$1{\cdot}\beta + x$	$2{\cdot}\beta + x$	$3{\cdot}\beta + x$	$4{\cdot}\beta + x$	$5{\cdot}\beta + x$	$6{\cdot}\beta + x$
	0	8	23	38	53	68	83	98
$NF^{(x)}$	d_1	10	25	40	55	70	85	100
	d_2	13	28	43	58	73	88	103
	$y = 2x$	$0{\cdot}\beta + 2x$	$1{\cdot}\beta + 2x$	$2{\cdot}\beta + 2x$	$3{\cdot}\beta + 2x$	$4{\cdot}\beta + 2x$	$5{\cdot}\beta + 2x$	$6{\cdot}\beta + 2x$
	0	16	31	46	61	76	91	1
$NF^{(2x)}$	d_1	18	33	48	63	78	93	3
	d_2	21	36	51	66	81	96	6
	$y = 3x$	$0{\cdot}\beta + 3x$	$1{\cdot}\beta + 3x$	$2{\cdot}\beta + 3x$	$3{\cdot}\beta + 3x$	$4{\cdot}\beta + 3x$	$5{\cdot}\beta + 3x$	$6{\cdot}\beta + 3x$
	0	24	39	54	69	84	99	9
$NF^{(3x)}$	d_1	26	41	56	71	86	101	11
	d_2	29	44	59	74	89	104	14
	$y = 4x$	$0{\cdot}\beta + 4x$	$1{\cdot}\beta + 4x$	$2{\cdot}\beta + 4x$	$3{\cdot}\beta + 4x$	$4{\cdot}\beta + 4x$	$5{\cdot}\beta + 4x$	$6{\cdot}\beta + 4x$
	0	32	47	62	77	92	2	17
$NF^{(4x)}$	d_1	34	49	64	79	94	4	19
	d_2	37	52	67	82	97	7	22
	$y = 5x$	$0{\cdot}\beta + 5x$	$1{\cdot}\beta + 5x$	$2{\cdot}\beta + 5x$	$3{\cdot}\beta + 5x$	$4{\cdot}\beta + 5x$	$5{\cdot}\beta + 5x$	$6{\cdot}\beta + 5x$
	0	40	55	70	85	100	10	25
$NF^{(5x)}$	d_1	42	57	72	87	102	12	27
	d_2	45	60	75	90	0	15	30

$P^{((\lambda-1)x)}\}$, where x is the shifting distance between two consecutive members of *NPS*.

Example: Using irregular file allocation algorithm, let us find a noninterfering processor set for the previous example whereby a circulating memory with $T^* = 105$ words is connected with $T = 30$ processors and primitive file $PF = \{0, 2, 5\}$, that is, $d_1 = 2$, $d_2 = 5$, and $j = 3$. Since $\beta = \gcd(30, 105) = 15$, let us select $PF_d = \{0, 2, 4\}$, leading to $d = 2$, $g = \beta/j = 15/3 = 5$. However, since $\gcd(\lceil\beta/j\rceil, \beta) = \gcd(5, 15) = 5 \neq 1$, the modular equation $d = \lceil\beta/j\rceil x - \beta\cdot y$ has no integer solution x, y. Therefore, the maximal approximation, $PF_d = \{0, 2, 4\}$, has to be modified by selecting such j that $\gcd(\lceil\beta/j\rceil, \beta) = 1$. Suppose that $j = 4$; then $PF_d = \{0, 2, 4, 6\}$, $g = \lceil\beta/j\rceil = \lceil 15/4\rceil = 4$. Therefore $d = 4\cdot x - 15\cdot y$; or $2 = 4\cdot x - 15y$. This equation has solution $x = 8$, $y = 2$ because $2 = 4\cdot 8 - 15\cdot 2 = 32 - 30$. Therefore, we obtain processor displacement $x = 8$ for the irregular allocation with $PF = \{0, 2, 5\}$. Let us now find how many members will be in the *NPS* processing set. Since primitive file *PF* has $d_1 = 2$ and $d_2 = 5$ we find numbers g_1 and g_2 by constructing the following equations:

$$2 = 8\cdot g_1 - 15y_1 \qquad g_1 = 4,\ y_1 = 2$$

$$5 = 8\cdot g_2 - 15y_2 \qquad g_2 = 10,\ y_2 = 5.$$

Therefore, the noninterfering processor set, *NPS*, contains λ members where $\lambda = g_2 - g_1 = 10 - 4 = 6$, $NPS = \{P^{(0)}, P^{(1\cdot 8)}, P^{(2\cdot 8)}, P^{(3\cdot 8)}, P^{(4\cdot 8)}, P^{(5\cdot 8)}\}$.

E.3 Resume

As follows from the material of this section, irregular file allocation is entirely algorithmic because one can find both x and λ for the noninterfering processor set, $NPS = \{P^{(0)}, P^{(1x)}, \ldots, P^{((\lambda-1)x)}$, via the algorithmic procedures we have specified.

If common words are then eliminated entirely from all interfering files one obtains irregular file allocation with the least interference, when memory introduces a minimal number of dummy clocks in file accessing.

F. CONCLUSIONS: PARALLEL OPERATION OF SEVERAL CIRCULATING MEMORIES

This paper has shown that if T processors are connected with one circulating memory having T^* words, then the following cases of allocation can be distinguished:

1. Regular allocation, whereby a single memory M is accessed by $g(j)$ processors with the following processor addresses: 0, $1x$, $2x, \ldots,$ $(g(j) - 1)x$, where $g(j) = \lceil\gcd(T, T^*/j\rceil$ and the processor displacement x is the root of the modular equation $d = g(j)\cdot x - \beta\cdot y$. As was shown in the paper, all regular allocations (minimal, nonminimal divisible, and nonminimal nondivisible) are noninterfering, that is, circulating memory M is completely filled with information and loses no clock periods during each file access.
2. Irregular allocation, whereby a single memory M is accessed by λ processors, where $\lambda = \max(g_i - g_{i+1})$ and the addresses of participating processors are: 0, x, $2x, \ldots, (\lambda - 1)\cdot x$. The allocation described is the least interfering, since a minimal number of words of memory M are empty. Thus during memory circulation, a minimal number of clock periods are wasted.

Therefore, to provide either noninterfering or minimal interfering file allocation one must connect only $g(j)$ or λ processors with one circulating memory. The remaining processors are idle. To eliminate their idleness, it is necessary for a regular allocation to have *NM* circulating memories serving T processors, where $NM = T/g(j)$, and $g(j)$ processors are connected with one memory (Figure 6).

Similarly, one finds that the number of memories for irregular allocation is $NM = T/\lambda$. In Figure 6 we have two circulating memories, $M1$ and $M2$, storing job files for eight processors partitioned into two groups, $GR1$ and $GR2$, where $GR1$ includes processors P_0, P_2, P_4, and P_6, and $GR1$ includes P_1, P_3, P_5, and P_7.

Generally, the addresses of the processors connected with the same memory are 0, $1x$, $2x, \ldots, (g - 1)x$ and they are determined via the solution of the modular equation, $d = gx - \beta y$, where 0 address is assigned to the processor that fetches the first output word, w, of the memory M. Therefore to exclude intersection among any two groups of processors, G_1 and G_2, connected to two different circulating memories, M_1 and M_2, allocations should be selected that are characterized by the same displacement x and the same number, $g(j)$ or λ, of processors included in each group, G_I. In Figure 6, $x = 6$, $g(j) = 4$. Thus, group $GR1$ is formed of the noninterfering processor set $NPS_1 = \{0, 6\cdot 1, 6\cdot 2, 6\cdot 3\} = \{0, 6, 4, 2\}$ and group $GR2$ is formed of noninterfering processor set $CPS_2 = \{1, 1 + 6, 1 + 12, 1 + 18\} = \{1, 7, 5, 3\}$. In this case we will obtain a very effective utilization of processor and memory resources, and either no clock periods, or a minimal number of them, are wasted in each allocation.

Therefore, for regular files, one obtains a high-performance memory allocation in which each circulating memory is completely filled with information. In addition, each memory will fetch $g(j)$ files to $g(j)$ processors and T processors will be served by $T/g(j)$ circulating memories. The allocation is noninterfering.

For irregular files, a very high performance is also achieved, whereby in each memory a minimal number of cells is empty and each memory fetches files for λ processors. The number of circulating memories serving T processors will be T/λ. Since λ is greater than $g(j)$, performance of irregular allocations is not as good as for the regular ones. Therefore whenever possible it is preferable to use regular allocations.

REFERENCES

1. Barnett, I.A. *Elements of Number Theory.* Boston: Prindle, Webber and Schmidt, 1969.

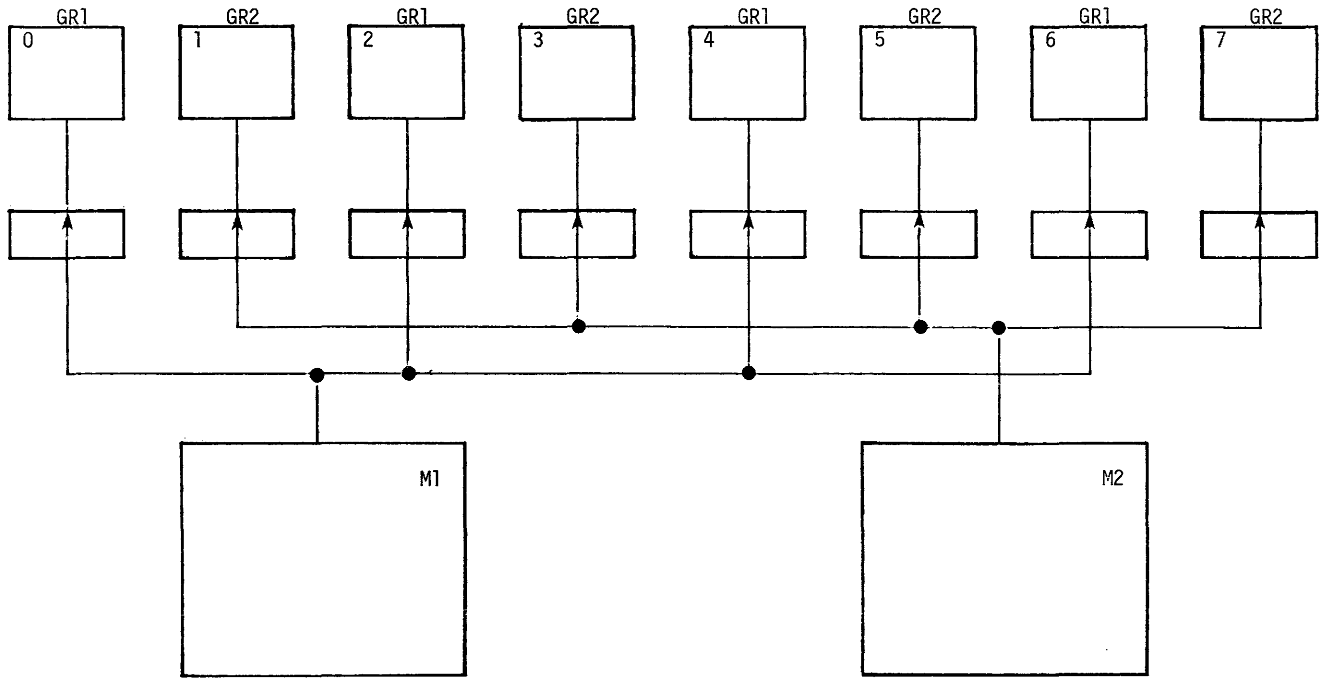


Figure 6—Concurrent operation of two circulating memories

2. Booth, T.L. *Sequential Machines and Automata Theory.* New York: John Wiley and Sons, 1967.
3. Golomb, S.W. *Shift-Register Sequences.* San Francisco: Holden-Day, 1967.
4. Kautz, W.H. (Ed.). *Linear Sequential Switching Circuits.* San Francisco: Holden-Day, 1965.
5. Elspas, B. "The Theory of Autonomous Linear Sequential Networks." *Transactions of the IRE,* CT-6 (1959), pp. 45–60.
6. Kartashev, S.P. "State Assignment for Realizing Modular Input-Free Sequential Logical Networks without Invertors." *Journal of Computer and System Sciences.* 7 (1973), pp. 522–542.
7. Kartashev, S.P. "Theory and Implementation of p-multiple Sequential Machines." *IEEE Transactions on Computers,* C-23 (1974), pp. 500–524.
8. Batcher, K.E. "The Multi-dimensional Access Memory in STARAN." *IEEE Transactions on Computers,* C-26 (1977), pp. 174–177.
9. Lawrie, D.H. "Access and Alignment of Data in an Array Processor." *IEEE Transactions on Computers,* C-24, (1975), pp. 1145–1151.
10. Siegel, H.J. "Controlling the Active/Inactive Status of SIMD Machine Processors." *Proceedings of the 1977 International Conference on Parallel Processing,* 1977, p. 183.
11. Siegel, H.J. "Interconnection Networks for SIMD Machines." *IEEE Computer,* 12, 6 (June 1979), pp. 57–65.
12. Lawrie, D., and C. Vora. (June 1979). "The Prime Memory System for Array Access." *IEEE Transactions on Computers,* C-31 (1982).

Reconfigurable fault-tolerant multicomputer network

by SVETLANA P. KARTASHEV
University of Nebraska, Lincoln
Lincoln, Nebraska
and
STEVEN I. KARTASHEV
Dynamic Computer Architecture, Inc.
Lincoln, Nebraska

ABSTRACT

This paper discusses fault-tolerant reconfigurations of a multicomputer network organized as a binary tree. The paper shows how to reconfigure a binary tree with faulty nonleaves with lost connectivity into a binary tree in which all faulty nodes become leaves or form more complex end-tree structures of higher dimension.

In both cases the faulty nodes are disconnected from a reconfigured fault-tolerant tree, which continues to function as a gracefully degraded tree made completely out of fault-free nodes.

The reconfiguration techniques developed are based on fine mathematical ideas of shift-register theory; they can be performed with only a single reconfiguration code (called a bias) that is sent concurrently to all fault-free nodes of a tree.

The techniques for finding this reconfiguration code are also very simple. For the case in which all faulty nodes become leaves, the required reconfiguration code can be found during the time of one mod-2 addition (one clock period). For the case in which all faulty nodes form an i-dimensional end-tree, the necessary reconfiguration code can be found following a simple process that includes (i-1) mod-2 additions performed sequentially.

Once the reconfiguration code is found, it is sent to all fault-free nodes of a binary tree. A fault-tolerant reconfiguration into a gracefully degraded tree with disconnected faulty nodes can be performed during the time of one clock period, since it includes the time of a one-bit shift and mod-2 addition.

A. INTRODUCTION

A very popular implementation for a distributed computing system is a binary tree. In this configuration each tree node is implemented as a separate and autonomous computational node.[1,2,3] The reason for this is that for conventional computations, the tree structure describes a variety of control and computational algorithms.

In general, control algorithms amenable to a tree implementation describe a tree hierarchical management structure whereby each higher-level node

1. manages and distributes information among all lower-level nodes that follow or succeed it in a tree, and
2. receives completion signals from all lower-level nodes indicating that all jobs assigned for execution have been completed.

Computational algorithms amenable to a tree implementation belong to a broad class of so-called *divide-and-conquer* algorithms (sorting and evaluation of arithmetic expressions, etc.), in which an entire algorithm can be easily partitioned into portions assigned to tree nodes for computations. Especially important is the use of binary trees in distributed databases, since most of the file accesses algorithms are based on queries organized as a binary tree.[4]

Trees have two types of nodes, *leaves* and *nonleaves*; here a leaf is understood as a node of the lowest level, $i = 0$, and a nonleaf node has level $i \geq 1$, where $i \leq n$ for the n-level tree. In this tree the node of the highest (n) level is called *the root*.

A multicomputer network that reconfigures into trees as well as other useful structures (stars and rings) can be organized if its nodes identified with computer elements (CE's) are interconnected with a special memory-processor bus (or DC-bus).[5,6] To organize a data broadcast between a pair of nodes N and N^* interconnected with the DC bus, it is sufficient for network node N to generate the position code or address of N^*.

Activation of the data path between N and N^* will be denoted as *transition* $N \rightarrow N^*$ meaning that

1. N will generate the position code or address of N^*.
2. N will establish a data path between N and N^*.
3. The data path between N and N^* can be made bidirectional; that is, it can run either from N to N^* or from N^* to N, and it can be any of the four types *PE-ME**, *PE-PE**, *ME-PE**, *ME-ME**—here the first element belongs to node N and the second element

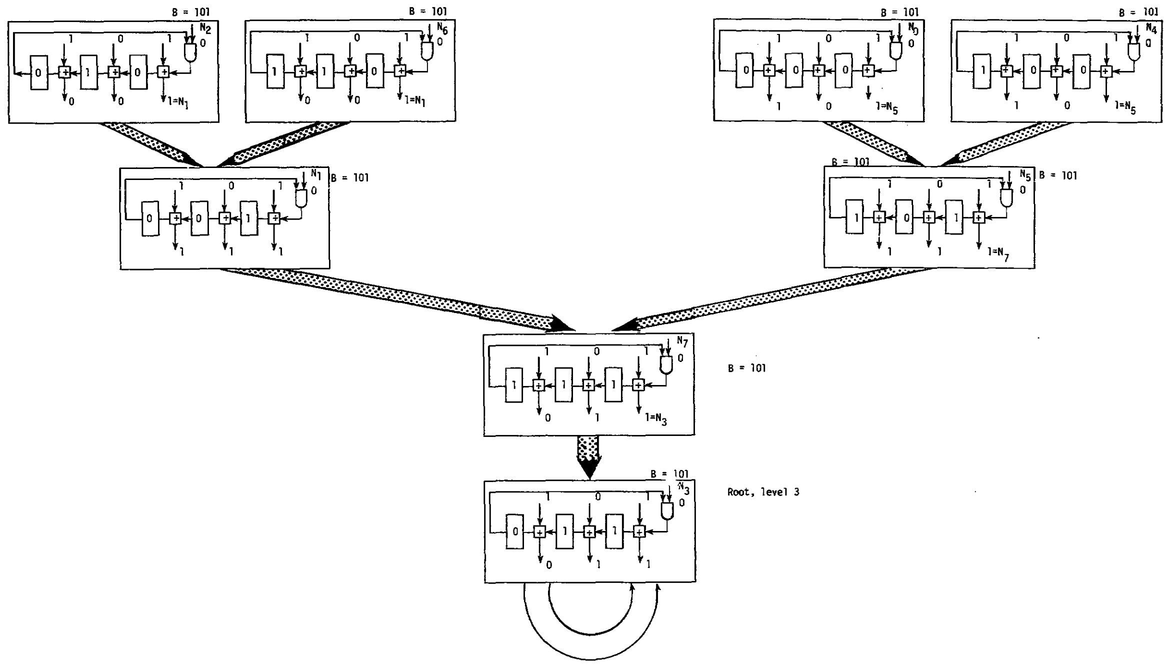


Figure 1—3-level tree

belongs to node N^*, and PE and ME stand for processor element and memory element.

To minimize the time of reconfiguration, it is reasonable to assume that for each network structure that can be assumed a rule of succession $N \rightarrow N^*$ will be maintained such that *during reconfiguration* each node N will have the minimal number of immediate successors N^*; this will minimize the reconfiguration time needed to establish the structure.

To hold to this rule during the reconfiguration of trees requires that the direction of succession be maintained from the leaves to the root. This will transform trees into *single-successor structures*, since each node N will have only one successor in this structure.

If all network transitions are established concurrently then the overall time of network reconfiguration will equal that of one transition.

A.1 Contribution of Shift-Register Theory

In this paper the following procedure will be used to generate the various tree structures that can be assumed by a distributed computing system. (This procedure is a particular case of a more general procedure that can generate not only trees, but also rings and stars.[6,7])

Assume that each tree node N is provided with a special shift-register that stores its position code N, which is of size n. (See Figure 1.) Suppose that in the given network structure to be assumed, node N should be connected with node N^* via a data path of type PE-PE^*, PE-ME^*, ME-ME^*, or ME-PE^*. Then for each type of communication between N and N^*, node N generates position code N^* using a left-shifted shift-register that generates N^* as follows:

$$N^* = 1[N]_0 + B,$$

where $1[N]_0$ is a 1-bit noncircular shift of N to the left and B is an n-bit reconfiguration constant brought with the reconfiguration instruction to *all network nodes that are requested for reconfiguration.* Reconfiguration constant B will be called *bias,* and the shift-register of Figure 1 is called a shift-register with the variable bias (SRVB). Since to generate a binary tree SRVB must always perform noncircular shift $1[N]_0$, subscript 0 will be omitted in this paper.

In Figure 1, the network of 8 nodes receiving bias $B = 101$ reconfigures into the 3-level tree with the root $R = 011$.

In all, since there exist 2^n different biases of size n, it is possible to generate 2^n different trees with an n-bit SRVB.

A.2 Usefulness of Various Tree Configurations for Improving Fault-Tolerance of Multicomputer Networks

The ability of the network to assume 2^n different tree configurations is important since it allows one to improve fault-tolerance in a tree.

Indeed, if in a binary tree k nonleaves become faulty, the tree may become unoperational by losing its connectivity and

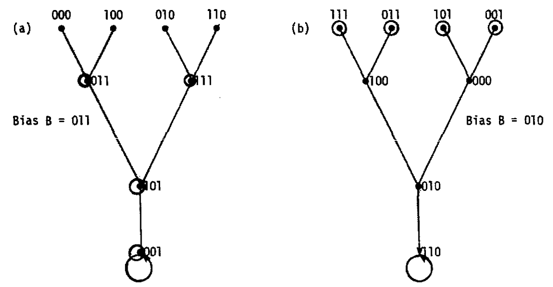


Figure 2—Fault-tolerant reconfiguration of a binary tree

being transformed into disjoint components. The fastest way of restoring connectivity in such a tree is to reconfigure as a new tree in which all faulty components become leaves; a tree with faulty leaves continues to function (although with some degradation in performance), since its connectivity is not lost.

For instance, consider two binary trees (Figure 2a, b) generated with 3-bit SRVBs receiving biases $B = 011$ and $B = 010$, respectively. The tree in Figure 2(a) has 4 faulty nodes (011, 111, 101, and 001). It cannot function, since with these nodes out of order it is transformed into 4 disjoint nodes. However, it can be reconfigured into a new tree generated with $B = 010$, which will be operational because in it all the faulty nodes have become leaves. To perform this reconfiguration requires a single instruction storing $B = 010$ that should be received by each fault-free node. The new tree structure is generated during one clock period, since to establish it each node should execute only two logical operations: 1-bit shift (one gate delay) and mod-2 addition (two gate delays). Thus reconfiguration into a new tree structure is an extremely efficient technique for rapidly restoring connectivity in this tree.

A.3 Monitor and Root Reconfigurations

In a distributed computing system organized as a tree, there are two approaches for performing tree reconfiguration:

1. monitor approach
2. root approach

A.3.1. Monitor approach. In the monitor approach, all fault-tolerant reconfigurations are performed by a system monitor that is connected with each tree node. The major advantage of this approach is that it can be performed very rapidly, since on finding out a status of each node the monitor, using the techniques presented in this paper, finds the best bias B that restores connectivity in the tree. Then it sends this bias to each tree node, thus effecting the tree reconfiguration into a new connected tree structure. Major disadvantages of this approach are

1. It requires that the monitor be connected with every tree node. This requires a complex interconnection network that performs these connections.
2. The monitor may become faulty, making the monitor approach totally inapplicable.

This requires consideration of the root approach.

A.3.2. Root approach. In the root approach, a fault-tolerant reconfiguration is performed by the root, the tree's connectivity having first been temporarily restored. However, this requires that some fault-free leaf nodes change their position codes (or addresses) and assume the position codes or vacancies created by all faulty nonleaves. Such node reassignment should be temporary—permanent reassignment requires extensive modifications in all program jump addresses for data exchanges with the nodes that have changed addresses.

Once the tree's connectivity is temporarily restored, its temporary root receives all information on the status of all tree nodes. Next, using the techniques presented in this paper, it finds the new tree's bias B and reconfigures this tree into one in which connectivity is restored for all fault-free nodes. Also, these nodes reassume the position codes that they had in the original or source tree.

The major advantage of this approach is that it is universal, that is, independent of the fault status of the monitor. Also, it requires no interconnections that are not available in the network. Its disadvantage is that it takes a longer time than the monitor approach, since it requires two steps: (1) temporary restoration of connectivity by changing nodes' addresses and (2) reconfiguration. Of these two, the first step is more time consuming than the second one, since to restore connectivity in a tree requires polling all successor nonleaves in a tree and assuming by leaves all the vacancies created by faulty nonleaves.

Therefore the monitor and the root approaches are complementary; the advantages of one approach are the disadvantages of the other. Thus both have their own areas of application.

As we shall show, both approaches involve tree reconfiguration in which faulty nonleaves become leaves. This requires proper selection of the bias B to accomplish this reconfiguration.

This paper is dedicated to describing a reconfiguration methodology aimed at selecting such a bias B that transforms the tree with faulty nonleaves into a connected fault-free structure requiring no node reassignment. If both leaves and nonleaves are faulty, then the reconfiguration is aimed at creating so-called *faulty end subtrees,* which are made entirely of faulty leaves and nonleaves. These subtrees can be disconnected from the overall tree without loss of connectivity in the tree.

The relationship of this paper with other work in the area is as follows:

1. Since it introduces original fault-tolerant reconfiguration techniques for distributed computing systems organized as binary trees, it is connected with other works on fault-tolerant reorganizations and reconfigurations.[8–12]
2. Since it uses the shift-register theory to develop reconfigurations of binary trees, it is connected with the literature on shift-register sequences.[13–19]
3. Since it discusses distributed computing systems organized as binary trees, it is associated with References 1 through 4.

A.4 Organization of This Paper

This paper is arranged as follows.

Section B introduces synthesis procedures for finding the root, R, once the bias B is given.

Section C finds an analytical expression for any tree node located on level i ($i = 0, \dots, n$).

Section D introduces types of reconfigurations in the tree; it shows that all possible reconfigurations are divided into two categories: (1) *level reconfigurations* and (2) *mixed reconfigurations,* where each mixed reconfiguration is a combination of several level reconfigurations.

Section E applies the interesting properties of level and mixed reconfigurations to improving fault tolerance in a tree. It introduces two cases of faulty nodes in a tree that will destroy its connectivity:

1. All faulty nodes are nonleaves; all leaves are fault-free.
2. Both leaves and nonleaves are faulty.

Next, Section E presents reconfiguration methodology aimed at finding the bias $B(r)$ of a reconfigured tree in which all fault-free nodes become connected. The tree thus reconfigured, composed of fault-free nodes, may function again, although its performance will be degraded by disconnection of all faulty nodes.

B. SYNTHESIS OF THE ROOT

Since in a binary tree the root may perform important fault-tolerant functions such as sending new bias to all tree nodes and informing all fault-free tree nodes about the faulty nodes, it is necessary to develop a simple synthetic technique for finding the root without actual construction of a binary tree.

Theorem 1, which follows, proposes this synthetic procedure. It is based on the following bias structure defined for a noncircular SRVB as follows: Given $B = GP_1 + \dots + GP_t$ where GP_i ($i = 1, \dots, t$) is its 1s position, otherwise called its *generating position.* Let $a_1, a_2, \dots, a_t$ be *bias distances* defined as follows: $GP_{i+1} = a_i[GP_i]$, that is, a_i shows the number of left-hand shifts between GP_i and GP_{i+1}, where i changes from 1 to $t = 1$. For GP_t, $a_t[GP_t] = 0$, since this is for a noncircular shift.

For each generating position, GP_i, let us form a mod-2 sum of all left j-bit shifts of GP_i ranging from $j = 0$ to $j = a_i - 1$, where a_i is the bias distance such that $a_i[GP_i] = GP_{i+1}$. Denote this sum by $BL(GP_i) = GP_i + 1[GP_i] + \dots + (a_i - 1)[GP_i]$. By construction, GP_i is an addend of $BL(GP_i)$ and GP_{i+1} is not an addend of $BL(GP_i)$.

With the introduction of BL sums we can establish the theorem that constructs the root.

Theorem 1. For a binary tree generated by an SRVB receiving bias $B = GP_1 + GP_2 + \dots + GP_t$, let GP_1, GP_3, $GP_5, \dots, GP_k$ be odd generating positions of the bias, where $k = t$ if t is odd and $k = t - 1$ if t is even. Form BL sums for odd generating positions of the bias as $BL(GP_1)$, $BL(GP_3), \ldots,$ $BL(GP_k)$. Then the root R is

$$R = BL(GP_1) + BL(GP_3) + \ldots + BL(GP_k).$$

For example, let us form the root R for the bias $B = 1\,0\,0\,1\,0\,1\,1\,0\,1\,0$—

$$R = BL(GP_1) + BL(GP_3) + BL(GP_5)$$

$$= \underbrace{\overset{GP_5}{1}}_{BL(GP_5)}\ 0\ 0\ \overset{GP_4}{0}\ \underbrace{\overset{GP_3}{1}\ 1}_{BL(GP_3)}\ \overset{GP_2}{0}\ \underbrace{\overset{GP_1}{1}\ 1}_{BL(GP_1)}\ 0$$

Find the successor N^* of R:

$$\begin{array}{rl} & 9\,8\,7\,6\,5\,4\,3\,2\,1\,0 \quad\ \ 9\,8\,7\,6\,5\,4\,3\,2\,1\,0 \\ N^* = 1[R] + B = & 0\,0\,0\,1\,1\,0\,1\,1\,0\,0 \ + 1\,0\,0\,1\,0\,1\,1\,0\,1\,0 \end{array}$$

$$\begin{array}{rl} & 9\,8\,7\,6\,5\,4\,3\,2\,1\,0 \\ = & 1\,0\,0\,0\,1\,1\,0\,1\,1\,0 = R. \end{array}$$

As we have seen, the technique for constructing the root is very simple. It can be easily performed by the programmer, who finds the root for the given bias B and then stores it in the instruction that reconfigures the network into the given single binary tree. Thus each tree node will receive the root and use it for forming codes of other tree nodes with which a given node wishes to communicate. This formation of the tree nodes from the root can be accomplished using the synthesis technique presented in the next section.

C. SYNTHESIS OF AN ARBITRARY TREE NODE (NONLEAF OR LEAF)

In this section we will develop a synthesis procedure for constructing a tree node of the ith level ($i = 0, \ldots, n$) if bias B is given. The node of the nth level (the root) was constructed in Section B. Here we present a more general result that extends our findings to any ith-level node ($i = n, n-1, n-2, \ldots, 0$). To establish this result, let us introduce the notion of *recursive sum*, $RS_i(k)$, where $k \leq i$, $i = n - j$, $j \leq n$.

Basis: $RS_o\ (0) = 0$, $i = n - n = 0$, $k = 0$.

Inductive step: $RS_i\ (k)$, where $i \geq 1$, $k \leq i$.

$RS_i\ (k) = X_i + RS_m\ (k-1)$, where $X_i = 2^{n-i}$, that is if SRVB stores $RS_i\ (k)$ then its position b_{n-i} always stores 1, and $k-1$ shows the number of other 1s positions that are necessarily more significant than X_i (Fig. 3); variable X_i will be called a *level variable.*

Once we have introduced the notion of recursive sum $RS_i(k)$ we can develop the technique for constructing the node of level $n - i$ ($i = n, n-1, \ldots, 0$).

Theorem 2. In a single binary tree formed with bias B, any tree node, $N(n-i)$, of level $n - i$ ($i = 0, 1, \ldots, n$) can be found as follows: $N(n-i) = R + RS_i\ (k)$ where $k \leq i$, where R is the root and $RS_i(k)$ is a recursive sum with level variable X_i.

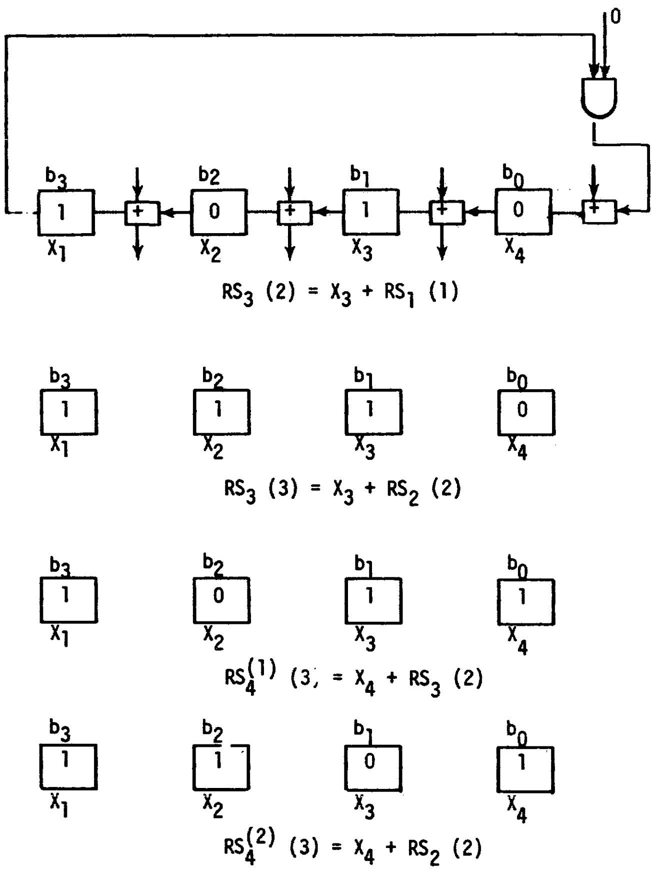


Figure 3—Recursive sums

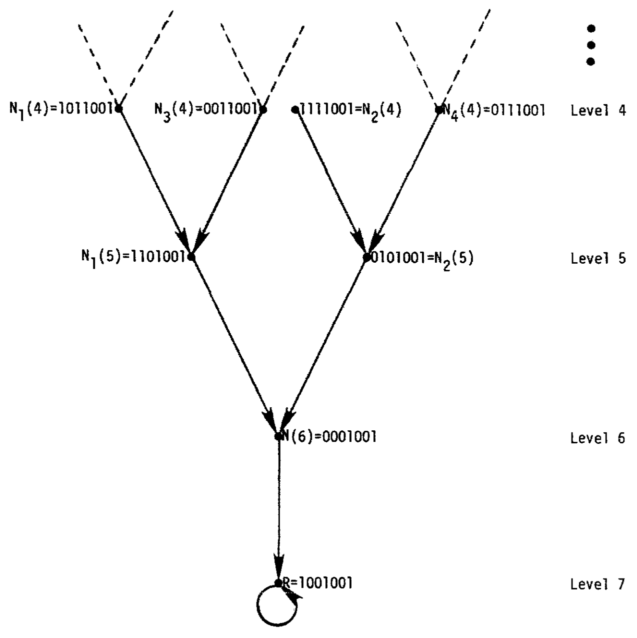


Figure 4—Synthesis of a tree node of level i

Example: Given bias B,

$$B = \overset{6}{1}\,\overset{5}{0}\,\overset{4}{1}\,\overset{3}{1}\,\overset{2}{0}\,\overset{1}{1}\,\overset{0}{1},$$

let us construct all the nodes of level $n - i = 7$, 6, 5, and 4. (See Fig. 4.) Here $n = 7$. Thus the root is of level 7: $R = BL(GP_1) + BL(GP_3) + BL(GP_5)$ and $GP_1 = 2^0 = 1$, $GP_3 = 2^3 = 8$, and $GP_5 = 2^6 = 64$. $BL(GP_1) = GP_1$ since $a_1 = 1$; $BL(GP_3) = GP_3$, since $a_3 = 1$ and $BL(GP_5) = GP_5$ since $a_5 = 1$.

Thus,

$$\begin{array}{cccccccc} & GP_5 & & & GP_3 & & & GP_1 \\ R = & 1 & 0 & 0 & 1 & 0 & 0 & '1. \end{array}$$

Indeed, the successor N^* of R is $N^* = 1[R] + B = 1[1001001] + 1011011 = 0010010 + 1011011 = 1001001 = R = X_1 + X_3 + X_7$.

The node of level 6 is: $N(n - 1) = R + RS_1(1) = R + X_1 = 0001001 = X_3 + X_7$.This node is succeeded by the root because $N^* = 1[N(6)] + B = 0010010 + 1011011 = 1001001 = R$.

There are two nodes of level $n - 2 = 5$: $N_1(5) = R + RS_2(1) = R + X_2 = 1101001 = 105$ and $N_2(5) = R + RS_2(2) = R + X_1 + X_2 = X_2 + X_3 + X_7 = 0110001$. Indeed, $N_1(5)$ is succeeded by $N^* = 1[N_2(5)] + B = 1010010 + 1011011 = 001001 = N(6) = 9$. Likewise $N_2(5)$ is succeeded by $N^* = 1[N_2(5)] + B = 1010010 + 1011011 = 0001001 = 9$, and so on. Similarly one can construct all the other nodes of this tree.

D. PROBLEM OF RECONFIGURATION

To solve the problem of reconfiguration in binary tree let us introduce one *recursive* and *successor-preserving* structure, P_i, of tree nodes generated with the bias B.

As will be shown in Section D.1, the P_i structure is recursive in the sense that the P_i structure for nodes of up to level $n - i$ can be obtained recursively from P_{i-1} already obtained for nodes of up to level $n - (i - 1) = n - i + 1$ by adding mod-2 a single level variable X_i to all nodes N' of P_{i-1}—

$$N(n - i) = X_i + N',$$

where $i = 1, 2, \ldots, n$. Further, the structure P_i is successor preserving, i.e. if two nodes N and N' are ordered, in P_i as $N > N'$ all their k-successors in the tree preserve this order. Here the k-successor of node N is the node $N^{(k)*}$ that is connected with the node N by the path of order k. (Thus the 1-successor is the immediate successor.)

Another important property of this node structure is that it is both *horizontal* and *vertical;* horizontal in that it orders all the nodes belonging to the same level $n - i$, vertical in that it orders all the nodes from tree levels n, $n - 1$, $n - 2, \ldots,$ $n - i$.

D.1 Recursive and Successor-Preserving Structure, P_i, of Tree Nodes Generated with SRVB

As we have seen, a node N of level $n - i$ is given as follows:

$$N(n - i) = R + RS_i\,(k),$$

where each $RS_i\,(k) = X_i + RS_m\,(k - 1)$, and X_i is the level variable. Thus all tree nodes belonging to the same level $n - i$ are identified with the same level variable $X_i = 2^{n-i}$, and a binary tree having n levels of nodes is characterized by n level variables $X_1, X_2, \ldots, X_n$.

Since recursive sum $RS_i\,(k) = X_i + RS_m\,(k - 1)$, each RS_i can be represented as follows using level variables $X_1, X_2, \ldots,$ X_n:

$$RS_i = a_1{\cdot}X_1 + a_2{\cdot}X_2 + \ldots + a_{i-1}{\cdot}X_{i-1} + X_i,$$

where $RS_0 \equiv 0$, $i \le n$, and $a_1, a_2, \ldots, a_{i-1}$ are binary coefficients. The binary coefficients $a_1, a_2, \ldots, a_i$ of each recursive sum RS_i are characterized as follows:

1. For each RS_i, $a_i \equiv 1$ and $a_{i+1} \equiv 0$.
2. All coefficients $a_1, \ldots, a_{i-1}$ for level variables $X_1, \ldots,$ $X_{i-1} > X_i$ can assume arbitrary values 0 or 1.

To establish the node ordering in each P_i, we assume that recursive sums are ordered as follows:

*P*1. $RS_i > RS_j$ iff $X_i > X_j$ where $X_i = 2^{n-i}$, $X_j = 2^{n-j}$
*P*2. $RS_i > RS_i{}'$ iff $RS_i + X_i > RS_i{}' + X_i$

Example: Consider $RS_1 = X_1$, $RS_4 = X_1 + X_3 + X_4$ and $RS_4{}' = X_1 + X_2 + X_4$. In accordance with $P1$, $RS_1 > RS_4$ and $RS_1 > RS_4{}'$ because $X_1 > X_4$. In accordance with $P2$, $RS_4{}' > RS_4$, because $RS_4{}' + X_4 = X_1 + X_2 = RS_2$; $RS_4 + X_4 = X_1 + X_3 = RS_3$ and $RS_2 > RS_3$.

With these preliminary observations, we introduce the P_i structure as follows.

Basis i = 1: $P_1 = (N_0, N_0 + X_1)$, where N_0 is an arbitrary tree node called *basis node;* X_1 is the level variable, $X_1 = 2^{n-1}$.

Inductive step $i \ge 1$: $P_i = (P_{i-1}, P_{i-1} + X_i)$, where P_{i-1} is the $(i - 1)$-dimensional structure, and $P_{i-1} + X_i$ is obtained from P_{i-1} by complementing the X_i variable in all nodes of P_{i-1}.

Example: Construct $P_i = (P_{i-1}, P_{i-1} + X_i)$ if $i \le 3$ and $P_1 = (R, R + X_1)$, where R is the root, that is $N_0 = R$; assume that the root R is generated by bias $B = 10001$. Using Theorem 1, $R = X_2 + X_3 + X_4 + X_5 = 01111$. (See Figure 5.)

$$P_1 = (R, R + X_1) = (01111, 11111)$$

$$P_2 = (\overline{P_1, P_1 + X_2}) = (\overline{R, R + X_1}, \overline{R + X_2, R + X_1 + X_2})$$

$$P_3 = (P_2, P_2 + X_3)$$

$$= (\overline{\overline{R, R + X_1}, \overline{R + X_2, R + X_1 + X_2}},$$

$$\overline{\overline{R + X_3, R + X_1 + X_3}, \overline{R + X_2 + X_3, R + X_1 + X_2 + X_3}})$$

here and throughout each bar is extended over nodes belonging to the same substructure.

D.1.1. Vertical and horizontal properties of the P_i structure. The P_i structure may describe both vertical and horizontal ordering among tree nodes depending on the selection of *basis*

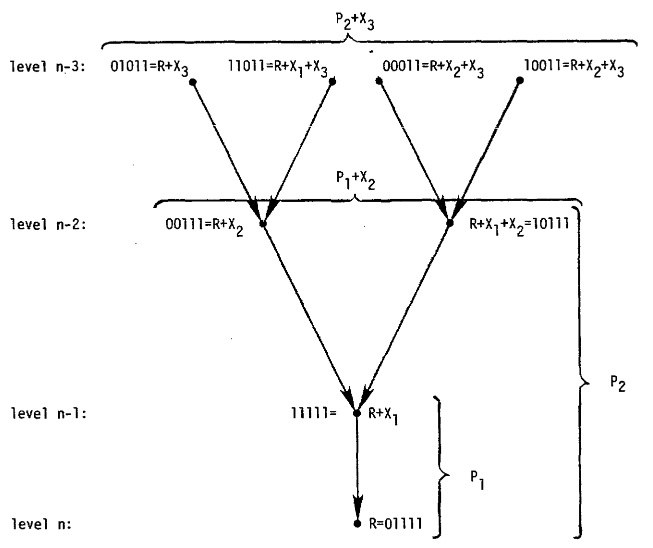


Figure 5—Recursive structures $P_1 = (N_0, N_0 + X_1)$, where $N_0 = R$; $P_2 = (P_1, P_1 + X_2)$ and $P_3 = (P_2, P_2 + X_3)$

node N_0 in basis structure $P_1 = (N_0, N_0 + X_1)$. If the basis node N_0 is the root R ($N_0 = R$) of the tree, then P_i describes the vertical ordering of the tree nodes belonging to levels n, $n-1, \ldots, n-i$ with respect to the root, which is considered the most significant node of the tree because $R = R + RS_0\ (0)$ and $RS_0 > RS_i$. If basis node $N_0 = R + X_i$, then P_{i-1} describes the horizontal ordering of the tree nodes belonging to the same level $n - i$. (See Figure 5).

D.1.2. Relative numbering of tree structures. Since in each P_i structure the position of a tree node is specified with respect to root R by the use of level variables $X_1, X_2, \ldots, X_i$, it is convenient to represent each P_i by the positions so specified. Namely, we assume that $R \equiv 0$ and $X_i \equiv i$. Then the relative structure of each P_i is given as follows:

$$P_0 = (\overline{0}),\ P_1 = (\overline{0}, \overline{1}),\ P_2 = (\overline{0,1}, \overline{2,12}),$$

$$P_3 = (\overline{\overline{0,1}, \overline{2,12}}, \overline{\overline{3,13}, \overline{23,123}})$$

$$P_4 = (\overline{\overline{\overline{0,1}, \overline{2,12}}, \overline{\overline{3,13}, \overline{23,123}}},$$
$$\overline{\overline{\overline{4,14}, \overline{24,124}}, \overline{\overline{34,134}, \overline{234,1234}}}),$$

and so forth, where each collection of integers $i_1, i_2, \ldots, i_k$ is understood as the node $N = R + X_{i_1} + X_{i_2} + \ldots + X_{i_k}$. For instance, $124 \equiv R + X_1 + X_2 + X_4$ where R is determined through bias B using the technique of Theorem 1. For instance, for $B = 1001$, $R = 0111 = X_2 + X_3 + X_4$. Thus

$$\begin{aligned} \text{node } 124 &\equiv R + X_1 + X_2 + X_4 \\ &= \cancel{X}_2 + X_3 + \cancel{X}_4 + X_1 + \cancel{X}_2 + \cancel{X}_4 \\ &= X_1 + X_3 = 1010, \end{aligned}$$

and so on.

D.2 Reconfigurations in a Binary Tree

As will be shown below, with the introduction of recursive structures $P_0, P_1, P_2, \ldots, P_k$, one can describe very easily all the reconfigurations that may occur in a binary tree. For each reconfiguration we will be dealing with two types of tree:

1. The *source tree TS* or original tree
2. The *reconfigured tree, TS (r)*

The nodes of source tree, *TS*, are called *source nodes*. The nodes of reconfigured tree are called *reconfigured nodes*.

Although reconfiguration into a new tree is performed with the reconfigured bias B, interesting structural properties of the tree reconfiguration are more easily understood if one introduces a special *map code MC* that maps each node N of the source tree TS into the reconfigured node $N' = N + MC$ of the reconfigured tree $TS\ (r)$. Thus the MC is conceived as the mod-2 sum of level variables that may change both the levels of tree nodes and positions of nodes within each P_i. Since $MC + MC = 0$, and 0 has no level variables, $R + MC$ becomes the new root, $R\ (r)$, in the reconfigured tree; $R\ (r) = R + MC$.

Therefore which reconfigurations can be performed over the given tree depends on what level variables are in MC.

To describe all the possible types of reconfigurations that can be performed, let us introduce the concepts of *level* and *mixed reconfigurations.*

D.2.1. Level reconfigurations. By the *X_i-level reconfiguration* or X_i reconfiguration we mean the reconfiguration determined by mapping code $MC = X_i$ ($i = 1, \ldots, n$). The X_i reconfiguration is performed on the level of all structures $P_i = (P_{i-1}, P_{i-1} + X_i)$ in the tree. It consists of exchanging P_{i-1} with $P_{i-1} + X_i$ as $P_{i-1} \rightleftarrows P_{i-1} + X_i$. Each reconfigured structure $P_i\ (r)$ obtained as a result of this is

$$P_i\ (r) = (P_{i-1} + X_i,\ P_{i-1}).$$

In a new tree $TS\ (r)$ the root $R\ (r)$ becomes: $R\ (r) = R + MC = R + X_i$. All source nodes of the level $n - i$ specified with level variable X_i become the reconfigured nodes of levels $n, n-1, n-2, \ldots, n-(i-1)$. On the other hand, all the source nodes of higher levels $n, n-1, \ldots, n-(i-1)$ become the reconfigured nodes of level $n - i$. Thus the change in levels occurs only for source nodes with levels $n, n-1, \ldots, n-i$. All source nodes of lower levels, $n-(i+1)$, $n-(i+2), \ldots, n-n = 0$, described by structures P_{i+1}, $P_{i+2}, \ldots, P_n$ retain their levels. Instead, in each P_{i+k}-structure each reconfiguration exchange is performed on the level of P_i, whereby each left element P_{i-1} of P_i is exchanged with its right element $P_{i-1} + X_i$.

Example: For the source tree of Figure 6(a) let us consider all the exchanges effected by X_1 reconfiguration. The reconfiguration is restricted to all P_1-pairs of this tree, which exchange their left and right components. This source tree has the following P_1 structures:

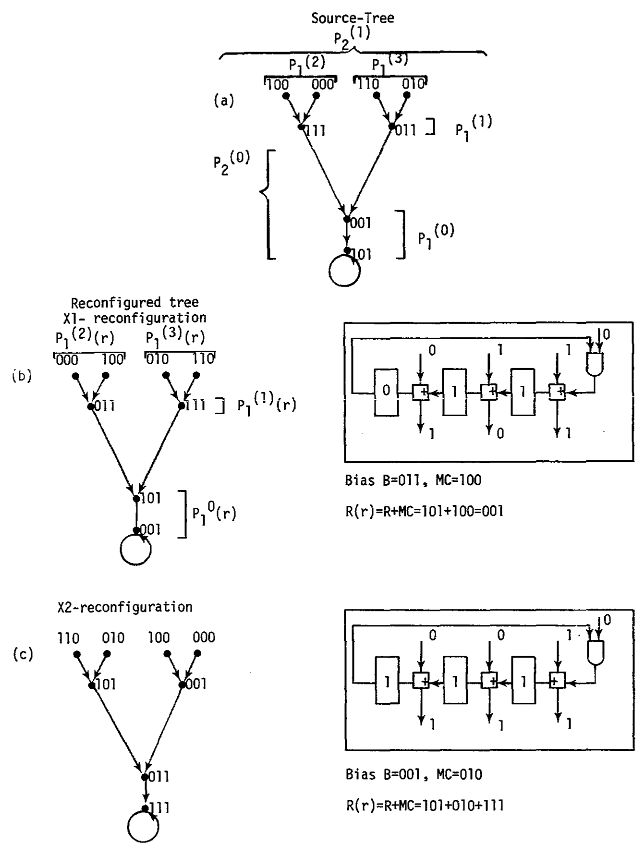


Figure 6—Level reconfigurations

Source P_1-structures	Reconfigured P_1 (r)
$P_1^{(0)} = (\overline{0},\overline{1}) = (101,001)$	$P_1^{(0)}$ $(r) = (\overline{1},\overline{0}) = (001,101)$
$P_1^{(1)} = (\overline{2},\overline{12}) = (111,011)$	$P_1^{(1)}$ $(r) = (\overline{12},\overline{2}) = (011,111)$
$P_1^{(2)} = (\overline{3},\overline{13}) = (100,000)$	$P_1^{(2)}$ $(r) = (\overline{13},\overline{3}) = (000,100)$
$P_1^{(3)} = (\overline{23},\overline{123}) = (110,010)$	$P_1^{(3)}$ $(r) = (\overline{123},\overline{23}) = (010,110)$

The reconfigured tree is shown in Figure 6(b). It has root $R(r) = 101 + X_1 = 001$ and is generated with reconfigured $B(r) = R(r) + 1[R(r)] = 001 + 010 = 011$. Indeed, $R(r)$ is succeeded as $1[R(r)] + B(r) = 010 + 011 = 001$. Reconfigured node $N = 101$ of level $n - 1$ is succeeded by $R(r)$;

$$1[101] + 011 = 010 + 011 = 001,$$

and so on. Similarly, one can find all the exchanges effected by X_2 reconfiguration and performed on the level of P_2-structures. The source tree has the following P_2-structures:

$$P_2^{(0)} = (\overline{0,1}, \overline{2,12}) \text{ and } P_2^{(1)} = (\overline{3,13}, \overline{23,123}).$$

The reconfigured $P_2(r)$ are

$$P_2^{(0)}\ (r) = (\overline{2,12}, \overline{0,1}) = (\overline{R + X_2, R + X_1 + X_2}, \overline{R, R + X_1})$$

$$= (\overline{111,011}, \overline{101,001})$$

$$P_2^{(1)}\ (r) = (\overline{23,123}, \overline{3,13})$$

$$= (\overline{R + X_2 + X_3, R + X_1 + X_2 + X_3}, \overline{R + X_3, R + X_1 + X_3})$$

$$= (\overline{101 + 011,101 + 111}, \overline{101 + 001,101 + 101})$$

$$= (\overline{110,010}, \overline{100,000}).$$

(See Figure 6c.) The X_2-reconfigured tree is shown in Fig. 6(c). As seen, its new root R $(r) = R + X_2 = 111$, which is generated with the following bias: $B(r) = R(r) + 1[R(r)] = 111 + 110 = 001$.

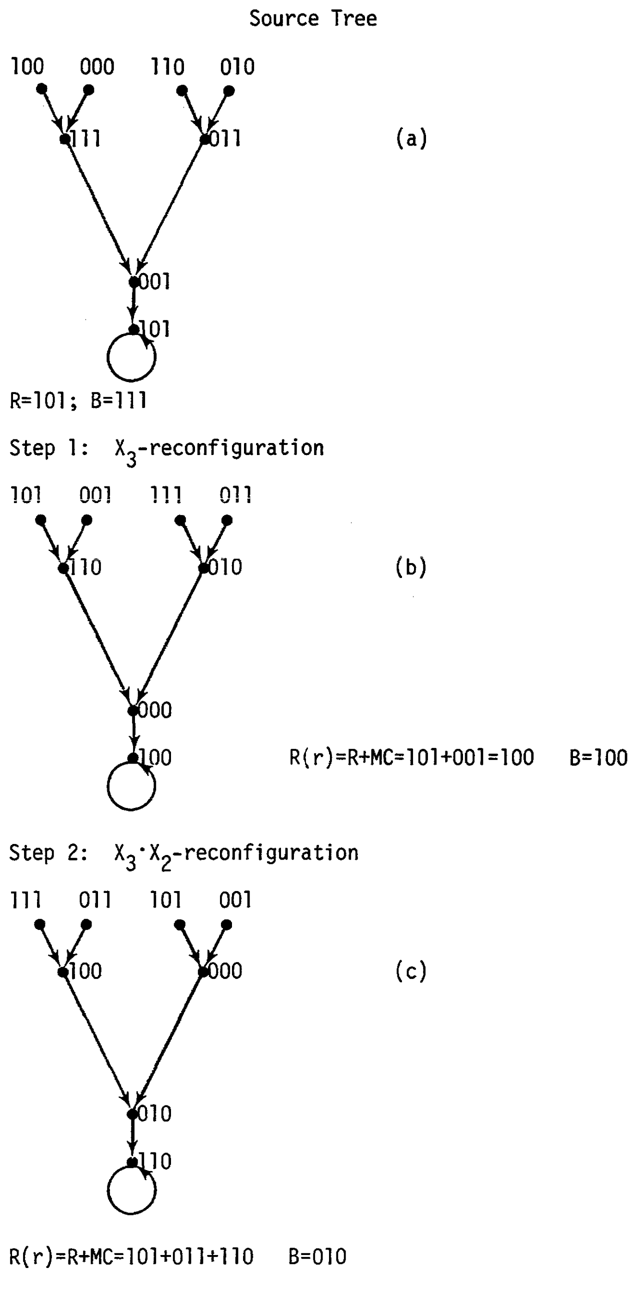


Figure 7—Mixed reconfigurations

D.2.2 Mixed reconfigurations. By mixed reconfigurations or $X_{j_1}X_{j_2}\ldots X_{j_k}$ reconfigurations we mean reconfigurations that are determined by mapping code $MC = X_{j_1} + X_{j_2} + \ldots + X_{j_k}$. The $X_{j_1}X_{j_2}\ldots X_{j_k}$ reconfigured tree can be found by applying sequentially $X_{j_1}, X_{j_2}, \ldots, X_{j_k}$-level reconfigurations to the source tree.

Example: For the source tree of Figure 7(a) let us apply the X_2X_3 reconfiguration using the mapping code $MC = X_2 + X_3$. Although this reconfiguration can be made in one step, it is interesting to perform it in two steps that perform 3, 32 reconfigurations respectively and to see that the tree obtained in two steps is coincident with the one obtained following application of $MC = 011$.

Step 1. Take the X_3 reconfiguration

$$\begin{aligned} P_3 &= (P_2, P_2 + X_3) \\ P_3(r) &= (P_2 + X_3, P_2) = (\overline{\overline{3,13}, \overline{23,123}}, \overline{\overline{0,1}, \overline{2,12}}) \\ R &= X_1 + X_3, MC = X_3 \\ R(r) &= R + X_3 = 100, B = 100. \end{aligned}$$

Thus

$$P_3(r) = (\overline{\overline{100,000}, \overline{110,010}}, \overline{\overline{101,001}, \overline{111,011}}).$$

Step 2. Let us form all P_2 structures for the X_3 reconfigured tree, since to perform X_3X_2 reconfiguration of the source tree it is sufficient to perform X_2 reconfiguration of the X_3 reconfigured tree of Figure 7(b).

$$P_2^{(0)} = (\overline{0,1}, \overline{2,12}) \rightarrow P_2^{(0)}(r) = (\overline{2,12}, \overline{0,1}).$$

Thus, since

$$P_2^{(0)} = (\overline{100,000}, \overline{110,010}), P_2^{(0)}(r) = (\overline{110,010}, \overline{100,000}).$$

The second

$$P_2^{(1)} = (\overline{3,13}, \overline{23,123})$$

is mapped into

$$P_2^{(1)}(r) = (\overline{23,123}, \overline{3,13}) = (\overline{111,011}, \overline{101,001})$$

E. FAULT-TOLERANT RECONFIGURATIONS

The interesting properties of level and mixed reconfigurations can be applied to faulty trees to restore connectivity in them. First of all, we note that if a tree has faulty leaf nodes only then no connectivity is lost, since no leaf has any predecessors in a tree. Therefore any loss of connectivity in a tree results from faulty nonleaves.

The material that follows is aimed at describing two cases that result in lost connectivity in a tree:

1. All faulty nodes are nonleaves. All leaves are fault-free.
2. Leaves and nonleaves are faulty.

E.1. Faulty Nonleaves and Fault-Free Leaves

If all leaves are fault-free, then to restore connectivity in a tree it is sufficient to perform either a level or a mixed reconfiguration that includes level variable X_n applicable to pair $P_n = (P_{n-1}, P_{n-1} + X_n)$. Since variable X_n exchanges the left P_{n-1} with the right $P_{n-1} + X_n$, the post-reconfiguration tree $TS(r)$ has the following changes in nodes positions: fault-free source leaves of $P_{n-1} + X_n$ become reconfigured nonleaves. Faulty source nonleaves of P_{n-1} become reconfigured leaves. Thus the entire tree connectivity is restored via reconfiguration, requiring no node reassignments.

Example: Consider a 3-level tree having root $R = 001$ and bias $B = 011$, as shown in Figure 8(a). Suppose that in this tree all nonleaves are faulty—that is, nodes 001, 101, 011 and 111 are faulty. The resulting network with lost connectivity is shown in Figure 8(b); it consists of four disjoint nodes. Restoration of connectivity may be accomplished with any reconfiguration that contains variable X_3, since X_3 exchanges P_2 and $P_2 + X_3$ within the P_3 structure, where

$$\begin{aligned} P_3 &= (P_2, P_2 + X_3) \\ P_2 &= (\overline{001,101}, \overline{011,111}), \\ P_2 + X_3 &= (\overline{000,100}, \overline{010,110}). \end{aligned}$$

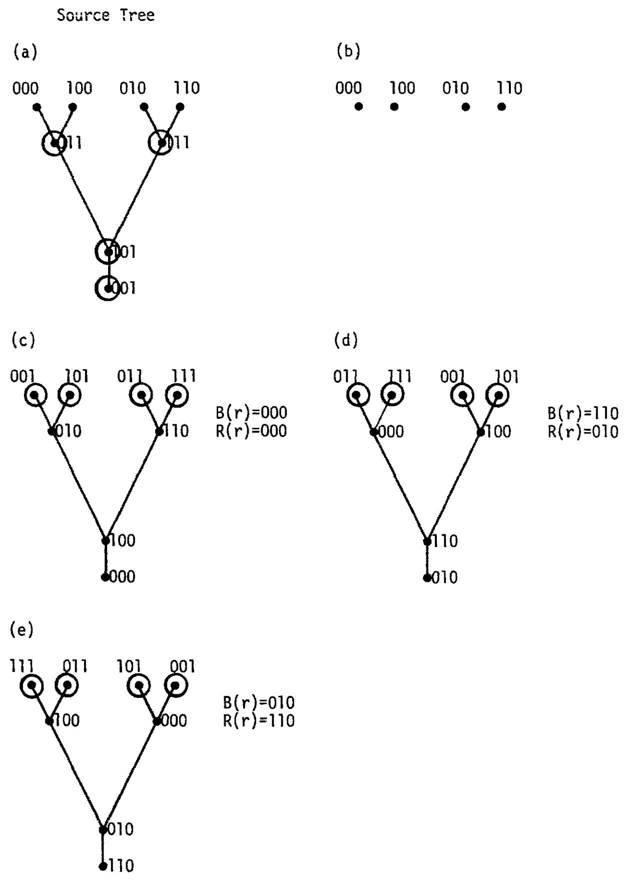


Figure 8—Obtaining fault-tolerant trees where faulty nodes become leaves

Thus the lost connectivity can be restored with reconfigurations identified with the mapping codes

$$MC_1 = X_3,\ MC_2 = X_2 + X_3, \text{ and } MC_3 = X_1 + X_2 + X_3.$$

If $MC_1 = X_3$, the reconfigured root is

$$R_1(r) = R + MC_1 = 001 + 001 = 000,$$

and the bias is

$$B_1(r) = 1[R(r)] + R(r) = 0.$$

(See Figure 8(c).) If $MC_2 = X_2 + X_3$,

$$R_2(r) = R + MC_2 = 001 + 011 = 010$$
$$B_2 = 1[R_2(r)] + R_2(r) = 100 + 010 = 110.$$

(See Figure 8(d).) If $MC_3 = X_1 + X_2 + X_3$,

$$R_3(r) = R + MC_3 = 001 + 111 = 110$$
$$B_3 = 1[R_3(r)] + R_3(r) = 100 + 110 = 010.$$

(See Figure 8(e).) As may be seen from the Figure, all these reconfigurations restore connectivity in the tree.

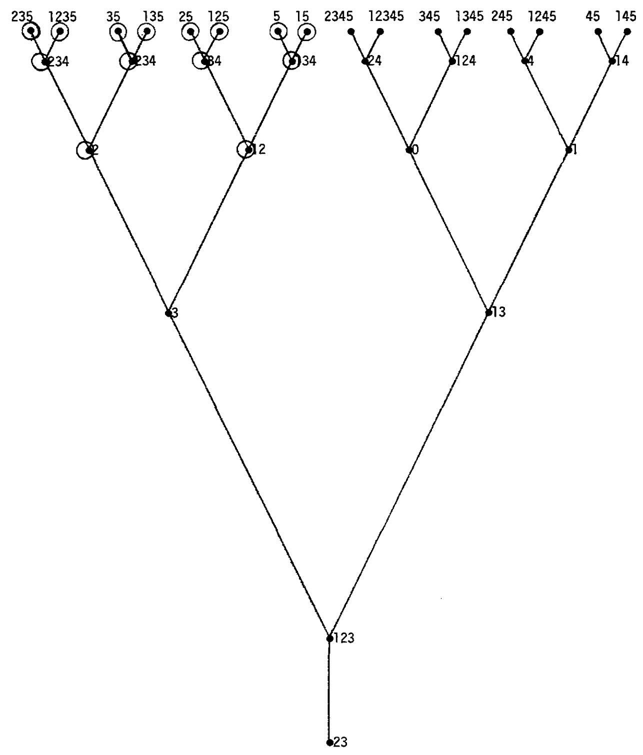


Figure 9—Fault-tolerant tree with faulty nodes forming two 2-dimensional end subtrees

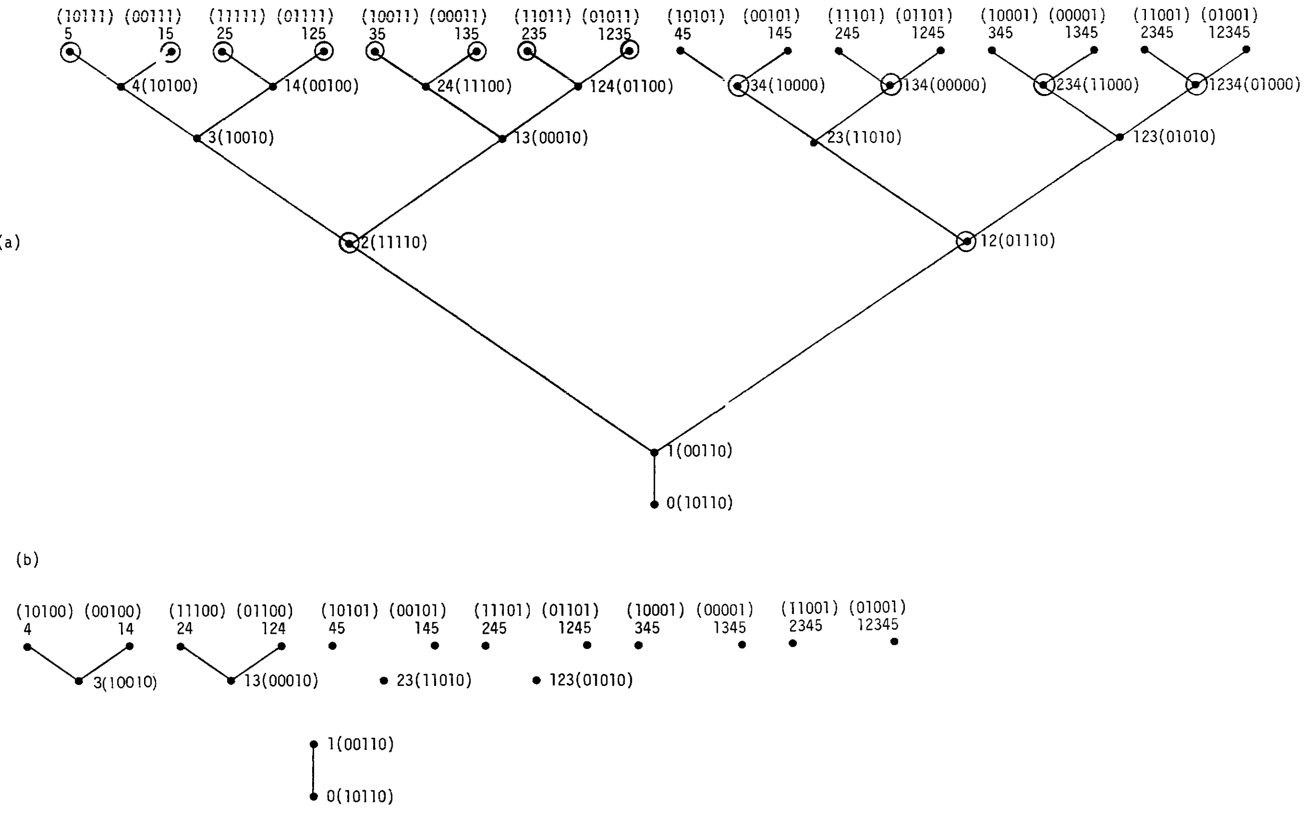


Figure 10—(a) source tree with faulty leaves and nonleaves; (b) disconnected components in this tree made of fault-free nodes

E.2. Faulty Leaves and Nonleaves

If in a tree faulty nodes are leaves and nonleaves, then the reconfiguration performed is described by the following steps.

1. Find the number of faulty leaves, $\#(L)$, and the number of faulty nonleaves, $\#(NL)$.
   If $\#(NL) > \#(L)$ perform any level or mixed reconfiguration having variable X_n.
   If $\#(NL) < \#(L)$, perform no X_n reconfiguration.
2. Reconfigure the source tree into a tree in which all faulty nodes are grouped into a so-called *end subtree.*

By the i-dimensional end subtree we mean a tree formed of structures P_i, $P_{i-1}, \ldots, P_0$, where P_i includes only leaves and P_{i-1}, $P_{i-2}, \ldots, P_0$ are all j-successors of P_i in this tree ($j = 1, 2, \ldots, i$). Since $P_0 = N^*$, P_0 is the unique i-successor of P_i. Figure 9 shows a tree with two 2-dimensional end subtrees formed entirely of faulty nodes; faulty nodes are circled. The entire reconfigured tree is given with its relative notation (introduced in Section D.1.2.) with respect to the root $R \equiv 0$ of the source tree, i.e. $R(r) = R + MC \equiv 0 + MC = 0 + X_2 + X_3 = 23$, where $MC = X_2 + X_3$.

Therefore, for the case of faulty leaves and nonleaves, one executes step 1 to reduce the number of faulty nonleaves, then selects a reconfiguration of the tree that groups all faulty nodes into one or several end subtrees. The effectiveness of such a reconfiguration is illustrated in the following example.

Example: A source tree with faulty leaves and nonleaves is shown in Figure 10(a) with its relative notation. As is seen from Figure 10(b), without reconfiguration the fault-free nodes of this tree function as disjoint components. Assume that this source tree is generated with source bias $B = 11010$. Using Theorem 1, we find the source root $R = 10110$, because $R = 1[R] + B = 01100 + 11010 = 10110$. In Figure 10, position codes of all tree nodes generated with $B = 11010$ are shown in brackets. As was said in Section D.1.2., each node code in brackets can be obtained by adding the root R with relative number of this node. For instance, node $N_{35} = X_3 + X_5$ has the following position code: $N_{35} = R + X_3 + X_5 = 10110 + 00101 = 10011$, and so on. As can be seen from this figure, this source tree has 8 faulty leaves and 6 faulty nonleaves. By step 1 above, this requires no X_5 reconfiguration. The best reconfigured tree is obtained with mapping code $MC = X_2 + X_3$. It gives reconfigured root $R(r) = R + MC = 10110 + 01100 = 11010$. This tree can be generated with bias $B(r) = 1[R(r)] + R(r) = 10100 + 11010 = 01110$; Figure 11(a) shows the fault-free, gracefully degraded tree and Figure 11(b) shows the generation of the $N_{24} \rightarrow N_0$ transition by the SRVB receiving $B(r)$, where $N_{24} = 11100$ and $N_0 = 10110$. As can be seen, the reconfigured tree is gracefully degraded and without faulty components. The latter are grouped into two end subtrees and disconnected. The complete reconfigured tree with faulty and fault-free components is shown in Figure 9. This reconfiguration takes only one clock period if one finds bias $B(r)$. The material below gives algorithmic procedures aimed at selecting $B(r)$.

E.2.1 Faulty End Subtree: Basis Step. The objective of the best reconfiguration is to assemble as many faulty nodes as possible into end subtrees that can be disconnected from the system.

The procedure for finding the maximal i-dimensional faulty end tree must start by finding a complete faulty structure P_i formed of leaves, where i is maximal. The next step is iterative. It consists of finding a complete faulty structure P_{i-1} somewhere in the tree and performing a tree reconfiguration that allows P_{i-1} to become the immediate successor of P_i. If

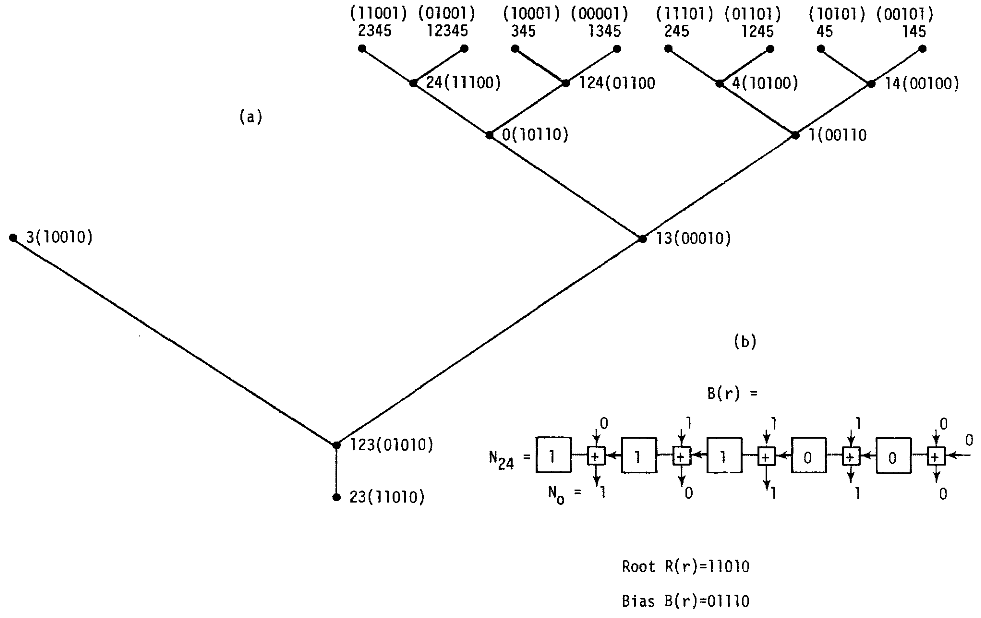


Figure 11—(a) gracefully degraded fault-tolerant tree with disconnected fault nodes; (b) generation of $N_{24} \rightarrow N_0$ transition in this tree with reconfigured bias $(B)r = 01110$

P_{i-1} is complete, the process is continued for P_{i-2}. If P_{i-1} is partial, the process stops. To perform this step, however, we must establish a so-called *closure of succession* between P_i and P_{i-1} in reconfigured tree $TS(r)$. This property is formulated as follows.

Theorem 3. For any two structures P_i and P_{i-1} in the source tree, where P_i is made of leaves and P_{i-1} is any other structure of dimension $i-1$ that does not succeed P_i in this tree, it is possible to find a bias $B(r)$ that generates a reconfigured tree $TS(r)$ in which P_{i-1} immediately succeeds P_i.

The following algorithm finds the bias $B(r)$ that allows P_{i-1} to immediately succeed P_i. Let N and N^* be two arbitrary nodes from P_i and P_{i-1} respectively. Form the succession equation

$$N^* = 1[N] + B(r)$$
$$B(r) = N^* + 1[N].$$

Example: Consider the 5-level source tree in Figure 12 given by the relative notation of its nodes with respect to root R. In this tree there are two complete faulty pairs, $P_2 = (5,15,25,125)$ and $P_1 = (3, 13)$, where P_1 does not succeed immediately P_2. Find the bias $B(r)$ that allows P_1 to be the immediate successor of P_2. Select arbitrarily $N \in P_2$ and $N^* \in P_1$;

$$N_5 = R + X_5,\ N_{13}{}^* = R + X_1 + X_3.$$
$$\begin{aligned} B(r) &= N_{13}{}^* + 1[N_5] \\ &= R + X_1 + X_3 + 1[R + X_5] \\ &= R + X_1 + X_3 + 1[R] + 1[X_5] \\ &= R + 1[R] + X_1 + X_3 + X_4, \end{aligned}$$

because $1[X_5] = X_4$. Since for the source $R = 1[R] + B$, source bias $B = R + 1[R]$. Therefore reconfigured bias $B(r) = B + X_1 + X_3 + X_4$. Thus by selecting any bias B of the source tree we can find the bias $B(r)$ of the reconfigured tree in which P_2 is made of faulty leaves and is immediately succeeded by P_1.

Assume that source bias $B = 10111 = X_1 + X_3 + X_4 + X_5$, giving source root $R = 01101$. Then reconfigured bias $B(r) = B + X_1 + X_3 + X_4 = X_5 = 00001$ and the reconfigured root $R(r) = 11111$. This reconfigured tree is shown in Figure 13(a). It differs from the source tree by $MC = R + R(r) = 01101 + 11111 = 10010 = X_1 + X_4$. Each node is given by its relative notation with respect to the source root and by its absolute codes generated by $B(r)$. As can be seen, all faulty nodes are grouped into two 1-end subtrees, requiring no node reassignment. Figure 13(b) shows how the SRVB that stores faulty $N_{125} = R + X_1 + X_2 + X_5 = 01101 + 11001 = 10100$ and receives $B(r) = 00001$ generates its faulty successor N_3 in this tree.

E.2.2. Faulty end subtree: iterative step. Since Theorem 3 establishes the succession closure between P_i and P_{i-1} in a reconfigured tree generated with bias $B(r)$, and two nodes $N \in P_i$ and $N^* \in P_{i-1}$ were selected arbitrarily for finding $B(r)$, we will have 2^{i-1} different reconfigured trees generated with as many different biases that maintain the succession between P_i and P_{i-1}.

These biases will form the allowable bias set $ABS = \{B_0(r), \ldots, B_{2^{i-1}-1}\}$ such that any $B_j(r) \in ABS$ generates a tree in which the leaf-faulty structure P_i is immediately succeeded by faulty P_{i-1}. This set ABS can be found by fixing node $N \in P_i$ and varying node $N^* \in P_{i-1}$.

Therefore, if there are two arbitrary complete faulty structures P_i and P_{i-1} in a tree where P_{i-1} does not necessarily succeed P_i and P_i is not a leaf structure

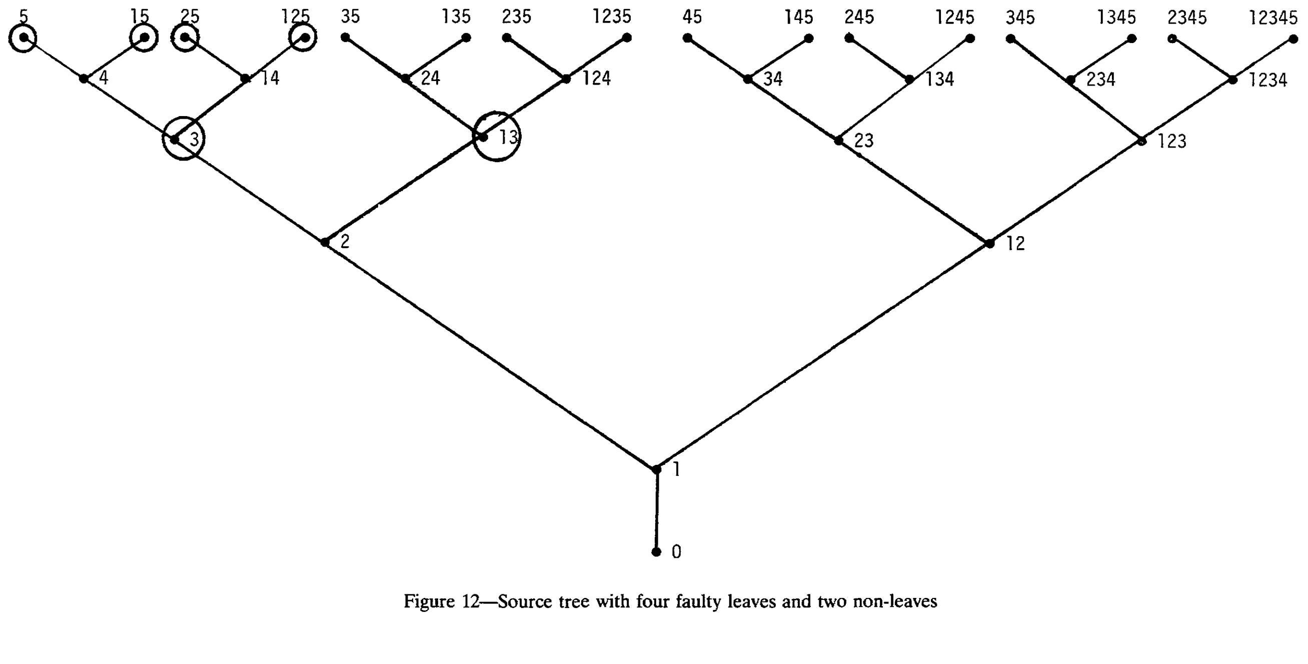


Figure 12—Source tree with four faulty leaves and two non-leaves

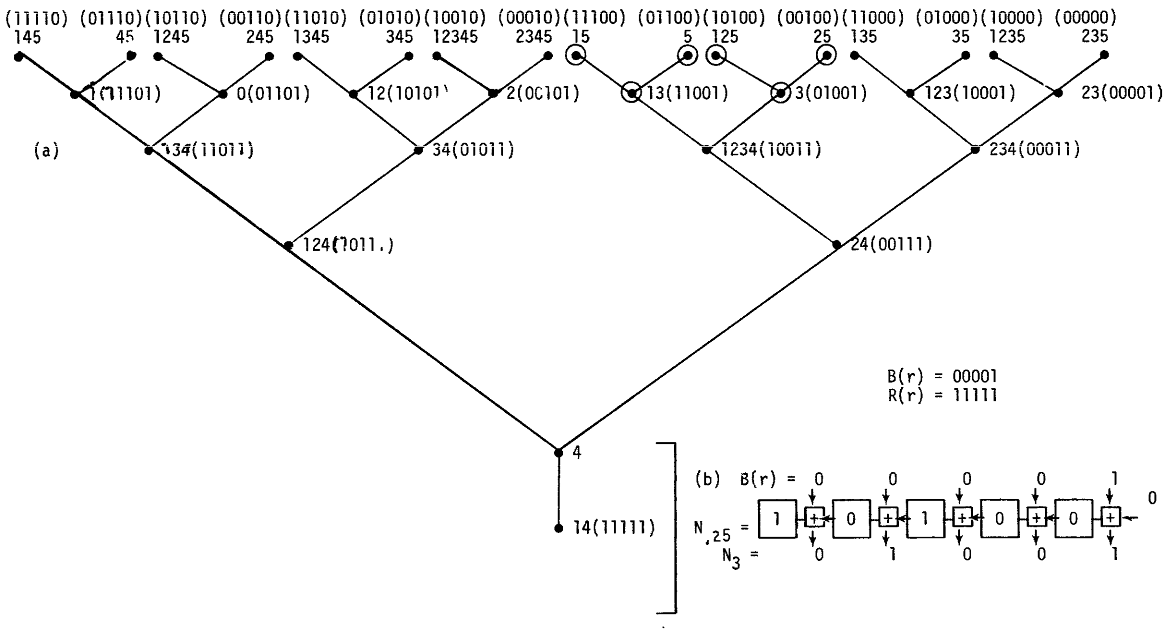


Figure 13—(a) gracefully degraded fault-tolerant tree; (b) generation of $N_{125} \rightarrow N_3$ transition in this tree

1. We can always perform X_n- reconfiguration to make P_i a faulty leaf pair.
2. We can always perform a reconfiguration whereby the leaf-faulty structure P_i is succeeded by faulty P_{i-1}.

An enlargement of the end subtree with the next faulty structure P_{i-2} depends on the location of faulty P_{i-2} in the tree, because a reconfigured tree in which P_{i-2} succeeds P_{i-1} must be such that the established succession between P_i and P_{i-1} has not been violated.

The following important property of allowable biases simplifies selection of the reconfigured bias that maintains succession between P_i, P_{i-1}, and P_{i-2}. Let $B_1(r)$ and $B_2(r)$ be two biases that maintain succession between P_i and P_{i-1}; if bias $B_1(r)$ is allowable, $B_2(r)$ is also allowable only if $B_1(r) + B_2(r)$ includes only a combination of level variables $X_1, \ldots, X_{i-1}$, of structure P_{i-1} where P_{i-1} succeeds P_i. Otherwise $B_2(r)$ is not allowable.

Therefore, to find the allowable bias $B_2(r)$ that maintains the succession between P_i, P_{i-1}, and P_{i-2} we have to find a candidate $B_2'(r)$ that maintains succession between P_{i-1} and P_{i-2}; it is understood that during the previous iterative step we found $B_1(r)$ that maintains succession between P_i and P_{i-1}. Next, we have to find $B_1(r) + B_2'(r) = S$. If S contains only the level variables $X_1, \ldots, X_{i-1}$ that specify structure P_{i-1}, $B_2'(r)$ is allowable bias, that is, $B_2'(r) = B_2(r)$. Thus there exists a reconfigured tree generated by $B_2(r)$ in which the faulty end subtree includes P_i, P_{i-1}, and P_{i-2}. If S contains other level variables, $B_2'(r)$ is not an allowable bias and no allowable $B_2(r)$ can be found. Thus the iterative process stops.

Example: Consider the binary tree with faulty nodes shown in Figure 10(a), in which there are the following faulty structures:

$$P_3 = (\overline{\overline{5,15}}, \overline{\overline{25,125}}, \overline{\overline{35,135}}, \overline{\overline{235,1235}})$$

$$P_2 = (\overline{34,134}, \overline{234,1234})$$

$$P_1 = (\overline{2}, \overline{12})$$

$$P_0 = (1)$$

For this tree, P_2 does not succeed P_3, and P_1 does not succeed P_2. Let us find a reconfiguration of this tree that forms the faulty end-tree of maximal dimension.

For the basic step, find the bias $B_1(r)$ that maintains succession between P_3 and P_2. Select $N_5 = R + X_5$ from P_3 and $N_{34} = R + X_3 + X_4$ from P_2. Form a successor equation that allows N_5 to be succeeded by N_{34}—

$$\begin{aligned} N_{34} &= 1[N_5] + B_1(r) \\ B_1(r) &= 1[N_5] + N_{34} \\ &= 1[R + X_5] + R + X_3 + X_4 \\ &= 1[R] + X_4 + R + X_3 + X_4 \\ &= 1[R] + R + X_3 \\ &= B + X_3. \end{aligned}$$

Thus if source bias $B = 11010$, $B_1(r) = B + X_3 = 11110$.

The iterative step proceeds as follows.

1. In the first iteration, let us attempt to include P_1 in the end-tree that includes P_3 and P_2. Assume that N_{34} is succeeded by N_{12}. Find $B_2(r)$ that allows such succession—

$$\begin{aligned} N_{12} &= 1[N_{34}] + B_2(r) \\ &= 1[R + X_3 + X_4] + B_2(r) \\ &= R + X_1 + X_2; \\ B_2(r) &= 1[R] + X_2 + X_3 + R + X_2 + X_1 \\ &= B + X_3 + X_1. \end{aligned}$$

Find

$$B_1(r) + B_2(r) = B + X_3 + B + X_3 + X_1 = X_1.$$

Since X_1 belongs to the set of level variables $[X_1, X_2]$ that specifies P_2, $B_2(r) = B + X_1 + X_3$ is an allowable bias that maintains the succession between P_3, P_2, and P_1. Thus $B_2(r) = B + X_1 + X_3$ generates a tree in which P_3, P_2, and P_1 will be included in the faulty end-tree.

2. In the second iteration, let us try to include $P_0 = (1)$ in the end-tree that includes P_3, P_2, and P_1. Find bias $B_3(r)$ that maintains succession between P_1 and P_0—

$$\begin{aligned} B_3(r) &= 1[N_2] + N_1 \\ &= 1[R + X_2] + R + X_1 \\ &= B + X_1 + X_1 \\ &= B. \end{aligned}$$

Find

$$\begin{aligned} S &= B_3(r) + B_2(r) \\ &= B + X_1 + X_3 + B \\ &= X_1 + X_3. \end{aligned}$$

Of these two variables, X_1 is allowable since it is included in P_1; X_3 is not allowable since it is not included in P_1, P_1 being characterized by $\{X_1\}$. Thus the iterative process stops and P_0 cannot be included in the end-tree that consists of P_3, P_2, and P_1. The bias that accomplishes the tree reconfiguration in which P_3, P_2, and P_1 are grouped together is $B_2(r) = B + X_1 + X_3$, where B is the source bias. Since $B_2(r) = R_2(r) + 1[R_2(r)]$, where $R_2(r)$ is the root of this reconfigured tree, and $B = 1[R] + R$ where R is the root of the source tree, $R_2(r) + 1[R_2(r)] + R + 1[R] = X_1 + X_3$. Taking into account that $R_2(r) + R = MC$ where MC is the map code, one obtains $MC + 1[MC] = X_1 + X_3$. That leads to $MC = X_2 + X_3$ because $1[MC] = 1[X_2 + X_3] = X_1 + X_2$. Thus $MC + 1[MC] = X_2 + X_3 + X_1 + X_2 = X_1 + X_3$. The reconfigured tree generated with $B_2(r) = B + X_1 + X_3$ and the root $R_2(r) = R + MC = R + X_2 + X_3$ is shown in Figure 9. As seen in this tree, the maximum number of faulty nodes are grouped into the faulty end tree of maximum dimensions. The only faulty node that is not included in this tree is $N_1 = R + X_1$. This tree can be generated with $B_2(r) = B + X_1 + X_3$. If one assumes that the source bias $B = 11010$, then $B_2(r) = 11010 + 10100 = 01110$.

F. CONCLUSIONS

In this paper we have studied fault-tolerant reconfigurations in a multicomputer network organized as a reconfigurable binary tree. We have shown how to reconfigure a binary tree with faulty nonleaves and lost connectivity into a binary tree in which all faulty nodes become leaves or form a more complex end-tree structure of higher dimensions. In both cases, these faulty nodes may be safely disconnected from the reconfigured fault-tolerant tree, which continues to function as a gracefully degraded tree made completely out of fault-free nodes. For the case that all faulty nodes become leaves, bias $B(r)$ of the reconfigured tree is found very easily during a one-step logical operation. For the case that all faulty nodes form an i-dimensional end-tree, the bias $B(r)$ of the reconfigured tree can be found by a simple process that includes $i - 1$ mod-2 additions performed sequentially. Once the bias $B(r)$ of the reconfigured tree is found, reconfiguration into a fault-tolerant tree (in which all faulty nodes can be safely disconnected) can be performed during the time of one clock period, since to perform this reconfiguration requires (1) the reception of the same code $B(r)$ by all tree nodes and (b) the formation of the new successor equation $N^*(r) = 1[N] + B(r)$ in each fault-free node N. Such a formation includes only 2 logical operations: 1-bit shift and mod-2 addition. Thus this fault-tolerant reconfiguration can be performed during one clock period.

REFERENCES

1. Despain, A., and D. Patterson. "X-Tree: A Tree Structured Multi-Processor Computer Architecture." *Proceedings of the Fifth Annual Symposium on Computer Architecture,* 1978, pp. 144–150.
2. Paxer, Y., and M. Bozygit. "Variable Topology Multicomputer." *Proceedings of the Second Euromicro Symposium on Microprocessing and Microprogramming,* Venice, 1976, pp. 141–149.
3. Wittie, L.D., and A. M. van Tilborg. "MICROS, A Distributed Operating System for MICRONET, A Reconfigurable Network Computer." *IEEE Transactions on Computers,* C-29 (1980), pp. 1133–1144.
4. Dewitt, David J., and Dina Friedland. "Exploiting Parallelism for the Performance Enhancement of Non-numeric Applications." *AFIPS, Proceedings of the National Computer Conference* (Vol. 51), 1982, pp. 207–216.
5. Kartashev, S. I., and S. P. Kartashev, "Problems of Designing Supersystems with Dynamic Architectures." *IEEE Transactions on Computers,* C-29 (1980), pp. 1114–1132.
6. Davis, C., S.P. Kartashev, and S.I. Kartashev. "Reconfigurable Multicomputer Networks for Very Fast Real-time Applications." *AFIPS, Proceedings of the National Computer Conference.* (Vol. 51), 1982, pp. 167–185.
7. Kartashev, S.P., and S.I. Kartashev, "Reconfiguration of Dynamic Architecture into Multicomputer Networks." *Proceedings of the 1981 International Conference on Parallel Processing,* Belleaire, Michigan, 1981, pp. 133–141.
8. Garcia-Molina, H. "Elections in a Distributed Computing System." *IEEE Transactions on Computers,* C-31 (1982), pp. 48–60.
9. Garcia-Molina, H. "Performance of Update Algorithms for Replicated Data in a Distributed Database." Department of Computer Science, Stanford University, Stanford, Ca., Rep. STAN-CS-79-744, June 1979.
10. Lamport, L. "The Implementation of Reliable Distributed Systems." *Computer Networks,* 2 (1978), pp. 95–114.
11. Lampson, B.W., and H.E. Sturgis. "Crash Recovery in a Distributed Data Storage System," Xerox, PARC Rep. 1979.
12. Menasce, D.A., G.J. Popeck, and R. R. Muntz. "A Locking Protocol for Resource Coordination in Distributed Databases." *ACM Transactions on Database Systems,* 5 (1980), pp. 103–138.

13. Elspas, B. "The Theory of Autonomous Linear Sequential Networks."*IRE Transactions on Circuit Theory,* 1959, pp. 45–60.
14. Zierler, N. "Linear Recurring Sequencer", *Journal of SIAM,* 7(1) (1965), pp. 31–48.
15. Kautz, W.H., (Ed.). *Linear Sequential Switching Circuits.* San Francisco: Holden-Day, 1965.
16. Golomb, S.W. *Shift Register Sequences.* San Francisco: Holden-Day, 1967.
17. Booth, T.L. *Sequential Machines and Automata Theory.* New York: John Wiley, 1967.
18. Kartashev, S.P. "Theory and Implementation of p-Multiple Sequential Machines." *IEEE Transactions on Computers,* 1974, pp. 500–523.
19. Kartashev, S.P. "State Assignment for Realizing Modular Input-free Sequential Logical Networks Without Inverters." *Journal of Computer and System Sciences,* 7 (1973), pp. 522–542.

PIONEER DAY

The work of Howard Hathaway Aiken and his colleagues at the Harvard Computation Laboratory will be the focus of this year's Pioneer Day. Technical sessions will be held on the afternoon of Wednesday, May 18. A special slide show will be given in the exhibit area.

A keynote talk will open the technical sessions; they will end with a panel evaluation of Aiken's contributions. In between, five speakers will highlight and analyze significant aspects of Aiken's work.

The evolution of computer technology during the development and operation of four successive large-scale computing machines will be discussed. Paths will be traced from electromechanical to magnetic and solid-state technology, and from straight machine-language coding to early attempts to permit programming directly in algebraic notation.

Then, Aiken's interest in computation and data processing, his stimulation of computer work in Europe and his development of pioneering research and educational programs in computer science will be discussed.

Each day of NCC, the story of machine development and of other facets of Aiken's work will be illustrated by a slide show, given in a room just off the south lobby of the Convention Center.

TELECOMMUNICATIONS

Randy J. Pilc
American Bell, Inc.
Lincroft, New Jersey

The Telecommunications track is composed of eight sessions that address particular aspects of computer communication problems. Included are sessions dealing with new services currently being planned, under the topic of network services planning. Keeping ahead of customer requirements with a public network is a combination of engineering, statistics, guessing, and luck. Planning for three to 10 years ahead requires assumptions about what the network will be used for and guesses about what equipment customers will be buying at each end and how it will be used. Or is it the other way around? Does the availability of a service, however well planned, influence users' applications and buying? Experience indicates that each—the network service availability and the users' actual requirements—affects the other, but not in a predictable manner. This session looks at network planning from the viewpoint of three differing services, each with unique problems and perspectives on future requirements.

The likely effect of new technologies on network solutions is addressed by the topic of distributed processing. The explosive growth of personal computers in corporate environments is taking distributed processing well beyond the "decade of the mini" (1970–1980). New infrastructures for data processing and communications must be developed and implemented to insure orderly, coherent growth of the corporate information resource. Equipment suppliers must carefully plan integrated strategies to address these needs.

Networking (both local and intrafacility) becomes even more important in this scenario. It will be the key to *true* distributed processing, enabling professionals and executives to use their powerful work stations fully—to unlock corporate (and other) databases and to move information.

This session focuses on the emerging trends and issues in distributed processing. Topics include a discussion of important network architectures (like SNA), the integration of minis and micros into DDP nets, and user implementation strategies. The network service bureau concept, as embodied in AT&T's AIS/Net 1000 and IBM's Information Network, will also be examined as a significant DDP direction.

The topic of advances in computer communications networks discusses these advances, which are rapidly being made in both long-haul and local networking areas. In the long-haul area, better network control technology for routing, flow, and congestion control and network configuration management is being developed. Progress continues in this area with the introduction of refined algorithms for network control suitable for implementation. Refinements center on improved performance, algorithm efficiency, and supporting distributed operation.

Local networking emphasizes new high-speed transmission technology, which is designed to support local distributed processing as well as multimedia communications.

Recent activities in standards are addressed by the topic of protocols for computer communications. The theme of this session is open systems interconnection (OSI). Since 1977, work on the reference model for OSI has been going on at a feverish pace. The communication protocol standards developed according to the reference model concepts will have a significant impact on the communications industry. The reference model provides the framework for developing protocols for the interconnection of heterogeneous systems. The model consists of a seven-layer hierarchy in which each layer (except the lowest) builds on the capabilities of the layer below. The resultant capabilities of these layers are provided to users by the highest layer, called the application layer.

The three presentations in this session address the recently developed draft proposals in the areas of the session layer (layer 5) protocol and the network layer (layer 3) service, and the role of intelligent peripherals in systems architecture. These timely presentations reflect major milestones recently reached by the standards bodies.

The topic of network management discusses issues related to the management of network availability and security. Network management can be considered the process controlling the availability of application system functions to the end user; it is involved in both hardware and software. Its functions may be categorized into administrative and operational managements. Administrative management involves a long-term systems administration which may have very little need to access data from the network. Operational management, on the other hand, involves a second-by-second systems operation and heavily relies on the data extracted from the network.

The four papers presented cover both network administration and operational management functions for networks based on systems network architecture (SNA).

The network security topic discusses vulnerability to interception. Technology-based society demands efficient and reliable communication networks. The introduction of microwave communication links has significantly increased the vulnerability of voice and data communications to electronic interception. With the wide acceptance of automated office systems and the increasing use of communicating word processors and distributed data terminals, proprietary, sensitive, and personal-privacy information can easily be intercepted through passive eavesdropping with relatively inexpensive equipment and limited technical resources. This session looks at the problem in depth and reviews techniques for protecting facsimile, data, and voice communications.

The topic of perspectives in digital voice processing discusses voice processing for low-bit-rate coding, synthesis, and automatic recognition, which typically requires algorithms of substantial computational complexity. But explosive advances in microelectronics now support the needed complexity, and economical single-chip and chip-set voice processors are of strong interest. This session offers a sampling of current algorithm research and the device technology that underpins hardware implementation.

The topic of human voice communications with computers looks at the idea of people's conversing with machines on human terms, which has finally been accepted as a feasible alternative to the more traditional manual data collection techniques. Interactive voice conversations with machines are now an accepted approach, particularly for sophicticated remote intelligent terminals. This approach offers the promise of humanized, simple-to-use computer systems of the future. This session provides a tutorial discussion about the concept of speech recognition and machine-generated voice response and discusses the obvious and not-so-obvious advantages of these techniques.

APPLICATIONS

Lorette L. Cameron
ARCO Metals
Rolling Meadows, Illinois

In the Applications track a number of areas have been selected for presentation. Special applications of technology emphasize interesting and different uses of technology for direct application to systems problems. Videodisc, special microprocessors, and various office automation techniques are explored. The law as a special application is covered in "Computers and the Law," which discusses a number of facets of legal issues in systems contracts and applications. Another fast-growing applications area is CAD/CAM, also included in the track. Finally, there is a session on creative ideas for systems development and measurement. Directly related to applications development, these ideas should provide some thought-provoking material for future development effort.

A standard session protocol for open systems interconnection (OSI)

by CHARLES E. YOUNG
American Bell
Lincroft, New Jersey

ABSTRACT

Protocol and service standards governing the session layer of the seven-layer reference model for OSI are expected to be finalized in 1983. These standards are being developed jointly by the International Organization for Standardization (ISO) and the International Telegraph and Telephone Consultative Committee (CCITT) groups, and will have wide applicability in data communications. Their basic characteristics and the open issues now being resolved are described. A status update on this fast-moving project will be presented at NCC '83.

INTRODUCTION

The Open Systems Interconnection (OSI) Reference Model[1,2] describes seven independent layers of protocol for data communications among open systems. In general terms it defines the services that each layer is to provide for its users (entities in higher layers). Each layer uses services provided by the layer below it, and enhances those services by means of its own protocol to provide services to its users. The seven layers are

- Application layer
- Presentation layer
- Session layer
- Transport layer
- Network layer
- Data link layer
- Physical layer

Two companion standards are to be adopted for each layer, one defining the services provided by the layer and the other specifying the layer protocol. The service definition standards have no conformance requirements, and are intended mainly for use in guiding the protocol standardization efforts and for tutorial purposes. The protocol specification standards, however, are intended to specify all requirements necessary to permit different entities in the same layer, but usually in different end systems, to communicate in a fully compatible manner.

This article describes those service and protocol aspects of the session layer that are reasonably stable, as of October 1982. The services are described first, which helps in understanding what the protocol is being designed to achieve. Then the status of the protocol design efforts is described, followed by an outline of the more important open issues. An update report on the status of these open issues will be presented at NCC '83.

SESSION SERVICES

The session services currently being studied by ISO and CCITT[3] would permit two users to organize and control their dialogue through use of a session connection established for them by the Session Layer (the provider). Service primitives are defined for use in establishing, utilizing, and releasing session connections, and four general types are expected to be used. A confirmed service primitive consists of four information transfers: a request from User 1 to the provider; a resulting indication from the provider to User 2; a response from User 2 to the provider; and a confirmation from the provider back to User 1. A nonconfirmed service primitive includes only a request and an indication; a single-user service primitive includes only a request; and a provider-initiated service primitive includes only indications, one to each user.

The service primitives that are currently agreed to in ISO and CCITT and their purposes are listed below according to the session connection phases in which they would be used and their types.

Establishment Phase

S_CONNECT (confirmed) would be used to establish a session connection and to negotiate values and ranges of values for session parameters.

Data Exchange Phase

S_DATA (nonconfirmed) would be used to transfer units of user data called session service data units (SSDUs) in sequence.

S_EXPEDITED_DATA (nonconfirmed) would be used to transfer expedited SSDUs, which may bypass other SSDUs in transit.

S_QUARANTINE_DELIVER (single-user) would be used to release for delivery a set of SSDUs previously stored by the provider at the sending user's request.

S_QUARANTINE_CANCEL (single-user) would be used to cancel delivery of a set of SSDUs previously stored by the provider at the sending user's request.

S_EXCEPTION_REPORT (provider-initiated) would be used to report unusual conditions not specifically covered by other services.

S_TOKEN_GIVE (nonconfirmed) would be used to transfer ownership of one or more tokens. A token is a session service attribute limiting the authority to request certain services to the owner of the token.

S_TOKEN_PLEASE (nonconfirmed) would be used to request ownership transfer of specified tokens.

S_SYNC_MINOR (confirmed) would be used to insert a minor synchronization point in the stream of S_DATA primitives being sent from one user to the other. There are "explicit" and "potential" minor synchronization points, which would have different rules governing their responses and confirmations.

S_SYNC_MAJOR (confirmed) would be used to insert a major synchronization point in both streams of S_DATA primitives being sent in the two directions. All minor and major synchronization points are identified by a single set of serial numbers incremented by the provider.

S_RESYNCHRONIZE (confirmed) would be used to renegotiate the current synchronization point serial-number value and token ownerships. With a "restart" resynchronization, the new serial number may be set back, but no lower than the most recent major synchronization point. Alternatively, the users may agree to abandon the old serial-number sequence and assign any new serial-number value for future use.

S_RESELECT (confirmed) would be used to revise parameter values and/or token ownerships, within ranges previously set during connection establishment.

S_ACTIVITY (nonconfirmed) would be used to initiate a new association between two users or to resume a specific previously unfinished association.

Release Phase

S_RELEASE (confirmed) would be used to release a session connection in an orderly manner without data loss. The release may be negotiated between the users in certain cases.

S_U_ABORT (confirmed) would be used by a user to abort a session connection. Data in transit may be purged.

S_P_ABORT (provider-initiated) would be used by the provider to abort a session connection for reasons internal to the provider. Data in transit may be purged.

TABLE I—Service subsets

	Basic Subsets:				
Service Primitives:	Duplex	Half-Duplex	Inter-active	Synchronized	Asymmetric Synchronized
S_CONNECT	X	X	X	X	X
S_DATA	X	X	X	X	X
S_EXPEDITED_DATA	X		X		
S_QUARANTINE_DELIVER			X		
S_QUARANTINE_CANCEL			X		
S_EXCEPTION_REPORT	X	X	X		X
S_TOKEN_GIVE		X	X	X	X
S_TOKEN_PLEASE		X	X	X	X
S_SYNC_MINOR				X	X
S_SYNC_MAJOR				X	X
S_RESYNCHRONIZE				X	X
S_RESELECT					X
S_ACTIVITY				X	X
S_RELEASE	X	X	X	X	X
S_U_ABORT	X	X	X	X	X
S_P_ABORT	X	X	X	X	X
Tokens Available:					
send data		X	X*	X*	X*
synchronize minor				X	X
synchronize major				X	X
release session		X	X	X	

*Half-duplex mode only

CONTROL OF SERVICE USAGE

The specific, detailed definition of each service primitive, its particular parameters and their permitted values and ranges of values, which are currently being studied in CCITT and ISO, will place inherent limits on the usage of every service primitive. Numerous modifications in these items will continue to be made until the protocol development efforts are completed. The current status of these items is contained in the most recent Draft Session Service Definition.[3]

Two other mechanisms, service subsets and tokens, are also included in the basic session services for purposes of usage control.

Service subsets are defined to match specific types of session users' needs, which are expected to predominate OSI session service usage. Five service subsets (see Table I) are currently defined, based largely on the session service primitives permitted to be used, but also containing certain detailed differences in parameter usage rules. Implementations only intended to provide services for specific user types can thus be simplified.

Four tokens are provided for use in selected subsets, as shown in Table I. Service subset selection during connection establishment determines which tokens are "available" and which ones are "not available" for use throughout the session. The particular meanings of these two terms for each token are shown in Table II. Table III lists the service primitives that can only be initiated by a user owning specific tokens.

SESSION PROTOCOL

The session protocol[4] being developed will specify the rules for information exchanges between peer entities in the session layer. The units of information exchange are called session protocol data units (SPDUs). SPDUs will be used in various ways to transfer information in support of session service primitives. The relationships between SPDUs and the session service primitives are illustrated in Table IV, and the ways that

TABLE II—Token availability definitions

Token	Capabilities When Available	Capabilities When Not Available
send data	Half-duplex data transfers (S_DATA)	Duplex data transfers (S_DATA)
synchronize minor	Minor synchronization points permitted (S_SYNC_MINOR)	No minor synchronization points permitted
synchronize major	Major synchronization points permitted (S_SYNC_MAJOR)	No major synchronization points permitted
release session	Token assignment required to initiate orderly release (S_RELEASE)	Only non-negotiated orderly release permitted (S_RELEASE)

TABLE III—Token assignments required

Service Primitive To Be Initiated	Tokens Required
S_DATA	send data*
S_SYNC_MINOR	send data* and synchronize minor
S_SYNC_MAJOR	send data*, synchronize minor and synchronize major
S_ACTIVITY	send data* and synchronize major
S_RELEASE	send data* and release session*

*Required only if "available."

SPDUs are currently proposed to be used in support of each service primitive are listed below.

It is important to recognize that the functions performed in the session protocol for a given service primitive are only permitted to take place when the associated service primitive is permitted to be used. Thus, the service subset and token authorization structures for the session services are reflected directly into the protocol, with respect to the functions permitted.

SPDU Usage for Each Service Primitive

S_CONNECT. Connect (CN) conveys connection request information from one session entity to the other. The normal response is either accept (AC) or refuse (RF). CN and AC contain information in parameters, such as the proposed subset and initial token assignments, which are negotiated among the two users and the two session entities to establish the characteristics and bounds permitted throughout the remainder of the session connection. These and all other SPDUs can only be conveyed via a transport connection, which must be provided by the transport layer. A characteristic of the initial session protocol is that only the session entity which previously requested the transport connection is permitted to send a CN SPDU over it.

S_DATA. Data transfer (DT) conveys session user data during the data transfer phase, which begins for a session entity when it sends or receives a valid AC. The right to send DT at any moment is governed by the send data token in half-duplex operation and by additional restrictions associated with synchronization and reselection functions when they are in use. Session connection flow control is attained only by refusing to accept further data transfers, and is reflected to the session sender through lower-layer protocol mechanisms. This technique is called *backpressure.*

S_EXPEDITED_DATA. Expedited (EX) conveys limited amounts of expedited user information using the transport-expedited service. EX SPDUs may bypass DT SPDUs en-route to the receiving user.

TABLE IV—Session protocol data units (SPDUs) for each service primitive

	Session Protocol Data Units (SPDUs) Initiated By:	
Service Primitive	Service Request	Service Response
S_CONNECT	Connect (CN)	Accept (AC)
		Refuse (RF)
S_DATA	Data Transfer (DT)	—
S_EXPEDITED_DATA	Expedited (EX)	—
S_QUARANTINE_DELIVER	Data Transfer (DT)*	—
S_QUARANTINE_CANCEL	Cancel(CL)	—
S_EXCEPTION_REPORT	Exception (ER)**	—
S_TOKEN_GIVE	Give Tokens (GT)	—
	Data Transfer (DT)*	—
S_TOKEN_PLEASE	Please Tokens (PT)	—
S_SYNC_MINOR	Data Transfer (DT)*	Mark Confirmation (MC)
S_SYNC_MAJOR	Data Transfer (DT)*	Prepare (PR)
		Mark Confirmation (MC)
S_RESYNCHRONIZE	Prepare (PR)	Prepare (PR)
	Resynchronize (RS)	Resynchronize Acknowledgement (RA)
S_RESELECT	Reselect (RE)	Reselect Acknowledgement (RK)
S_ACTIVITY	Start Activity (SA)	—
	Continue Activity (CA)	—
S_RELEASE	Finish (FN)	Disconnect (DN)
		Not Finished (NF)
S_U_ABORT	Abort (AB)	Abort Accept (AA)
S_P_ABORT	Abort (AB)**	
	Abort Accept (AA)**	

* Service information is contained only in parameters of the SPDU.
** Provider initiated.

S_QUARANTINE_DELIVER. The end of quarantine parameter in DT is used to signal that data quarantined for delayed delivery, if any, should be released for delivery.

S_QUARANTINE_CANCEL. Cancel (CL) is used to signal that data quarantined for delayed delivery, if any, should be canceled.

S_EXCEPTION_REPORT. Exception (ER) is used to convey information concerning unusual conditions detected in the session layer, which should be reported to the distant session user.

S_TOKEN_GIVE. Either Give tokens (GT) or the give tokens parameter in DT (when there is data to send) may be used to signal transfer of tokens from the owning user to the other user.

S_TOKEN_PLEASE. Please tokens (PT) is used to signal a request to transfer tokens from the owning user to the requesting user.

S_SYNC_MINOR. Parameters in DT are used to signal a potential or explicit minor synchronization point between two SSDUs. Mark confirmation (MC) is used to acknowledge all minor synchronization points up to one identified by its serial number in a parameter of MC.

S_SYNC_MAJOR. Parameters in DT are used to signal a major synchronization point between two SSDUs. Prepare (PR), returned via the transport-expedited service, is transferred just prior to MC in response to receipt of a major synchronization point. PR marks a major synchronization point in the expedited-data path, and signals that the MC is being sent. MC marks the point in the return normal data flow of the major synchronization point, thus concluding the previous dialogue unit. (PR is only sent when the transport-expedited service is in use).

S_RESYNCHRONIZE. Resynchronization is initiated by transmission of PR, with a parameter specifying that resynchronize (RS) is being sent. RS is used to force resynchronization in the exchange of data to a specified synchronization point and to force renegotiation of token ownerships. Typically, resynchronization is utilized after a failure is detected. Resynchronize acknowledgment (RA), preceded by a PR message, is the normal response to RS. RA includes the agreed new token ownerships and the new serial number as parameters. Data may be lost as a result of the resynchronization function. (PR is only sent when the transport expedited service is in use).

S_RESELECT. Reselect (RE) is sent to initiate renegotiation of parameter values and/or token ownerships within ranges established during session connection establishment. The normal response is reselect acknowledgment (RK), completing the renegotiation.

S_ACTIVITY. Start activity (SA) signals the beginning of a new association between two session users and conveys the initial synchronization serial number for the association. It also can convey a limited amount of user data. If the activity type signaled in S_ACTIVITY is "resume," instead of "start," then continue activity (CA) is sent, which must also convey the identity of the previously unfinished user association.

S_RELEASE. Orderly release is initiated by finish (FN). The normal response is disconnect (DN); however, the receiving user is permitted to decline in certain cases when the release session token is available. Not finished (NF) may then be returned in response to FN.

S_U_ABORT and S_P_ABORT. Abort (AB) is used to initiate the abort procedure, with the source (user or session entity) and reason included as parameters. Abort accept (AA) is the proper response. Parameters in AB and AA may also be used to negotiate whether or not the same transport connection may be reused to establish a new session connection.

CURRENT PROTOCOL DESIGN STATUS

The session protocol for OSI is being designed taking into consideration two existing standards, CCITT Recommendation S.62[5] and ECMA-75.[6] Together these two standards will permit the session services needed for OSI to be provided in a manner that satisfies the identified needs of both CCITT and ISO. S.62, having been designed prior to the OSI reference model, specifies some items clearly belonging in layers above the session layer. These items will be handled by the session protocol as user data. Also, some of the session functions and many of the parameters are defined differently in S.62 and in ECMA-75. The major current effort is to reconcile these differences in a manner that satisfies the session service definition and requires the minimum amount of change in either standard.

Major Issues and Status

Several of the major issues identified initially have already been successfully resolved, as of October 1982. It has been agreed to use the S.62 encoding principles, and compatible release and abort procedures have been agreed on, including compatible rules governing reuse of an underlying transport connection after completion of a session connection. Most of the protocol data units in the two standards have been mapped together; however, there are a few basic issues remaining, which are currently being studied. Most of them involve synchronization, and the more significant issues currently identified are outlined as follows:

1. Should the S.62 window mechanism, which permits multipie minor synchronization points to remain outstanding, be controlled in the session protocol or left for control in the users' protocol? The alternatives result in different session protocol mappings.
2. Should the users, when employing the session synchronization services, be permitted to request services, such as reversal of control states governed by tokens, while outside of any user association known to the session layer? The alternative is to require the users to enter into a new activity before requesting such services, which results in different protocol mappings.
3. In S.62 there is a "master/slave" relationship in the protocol, whereas the ECMA-75 protocol is symmetrical. How should the master/slave relationship concept be integrated into the overall protocol (without simply having two different protocol machines)?

REFERENCES

1. "Information Processing Systems—Open Systems Interconnection—Basic Reference Model," ISO/Draft International Standard (DIS) 7498 (ISO/TC97/SC16 N890), Feb. 4, 1982, as revised by "Changes to DIS 7498," ISO/TC97/SC16 N1226, June 1982.
2. "Reference Model Of Open Systems Interconnection For CCITT Applications," COM VII—No. R9(C), Appendix 12 of Annex 8 to "Report Of The Working Party VII/5 Melbourne Meeting, March 1982."
3. "Draft Basic Connection-Oriented Session Service Definition," ISO/TC97/SC16 N1166, June 1982.
4. "Current Status of Draft Basic Connection-Oriented Session Protocol Specification," ISO/TC97/SC16 N1167, June 1982.
5. "Control Procedures For The Teletex Service," CCITT Recommendation S.62, *Yellow Book*, Fascicle VII.2, Geneva, 1980.
6. "Standard ECMA-75 Session Protocol," European Computer Manufacturers Association, January 1982.

The role of the Intelligent Peripheral Interface in systems architecture

By I. DAL ALLAN
Sperry Univac
Santa Clara, California

ABSTRACT

The American National Standards Institute (ANSI) work committees on computer interfaces are actively developing proposed standards that will affect almost all vendors of computers and peripherals. This paper reviews one project, the Intelligent Peripheral Interface (IPI), in particular, and references the application area of the Small Computer Systems Interface (SCSI).

The architecture of future computer systems is going to depend in large part on the types of interfaces used, because of the ever-increasing role that peripherals play in systems performance, cost, availability, and reliability. The capability of peripherals dictates system performance—more than the central processor does, in most configurations. The cost of peripherals represents more than half of what the user pays for a system.

The interface chosen to interconnect peripherals has a major influence on the cost and performance tradeoffs that a vendor can make. This paper addresses these issues with regard to the ANSI efforts on intelligent interfaces.

The Small Computer Systems Interface (SCSI) and the Intelligent Peripheral Interface (IPI) are being defined as proposed American National Standards Institute (ANSI) standards by the ANS committees X3T9.2 and X3T9.3, respectively. Every industry representative has a voice in how these interfaces are defined, because ANSI standards are defined by industry volunteers who attend the meetings and contribute.

In the U.S., both the definition of and compliance with an ANSI standard are voluntary, and usually the result of market pressures. This is not necessarily true overseas, where standards may be set by the governments, and compliance with them is a requirement for all vendors. The standards defined by ISO are often adopted in their entirety, so it is quite feasible for a voluntary ANSI standard to become an ISO standard and wind up as a regulated requirement in foreign markets.

The availability of microprocessors and large-scale integration (LSI), both commercial and custom, has changed the way in which systems can be configured. The inexpensive micro makes it feasible to incorporate logic that formerly resided in the controller into the device, and to place in the controller logic that formerly resided in the operating system. The definition of controller varies by market environment, and is commonly known by different designations, such as storage director (SD) on mainframes and the direct memory adapter (DMA) on minis.

There are three general definitions of interfaces—channel, control, and device. Usually, the three only occur as separate and discrete interfaces on large mainframe configurations. In most mini configurations such as that illustrated in Figure 1, the functions of the channel and control interfaces are integrated onto the system bus. The device interface (which is most commonly an SMD) connects directly to a DMA controller that shares the CPU bus with other system elements.

This type of implementation has made DMA controllers, and subsequently the processor complex, very sensitive to technology changes in devices. Either every new device has to act just like the previous, or it causes some level of change in the DMA controller. The control interface, which connects to a head of string (SC—string controller), is used to mask unique device dependencies.

The minicomputer world is turning toward the string concept (which has been much used by mainframes), because it addresses the general market desire for a device-independent interface. Figure 2 illustrates both a mainframe and a mini with heads of string. The head of string may stand alone or be integrated into the first device, with others attached to it via a device interface. Alternatively, each device may have an integrated head of string so that there is a one-to-one relationship.

In covering the application areas for a peripheral interface it is first necessary to look at the characteristics of each of the three types of interface:

1. Device interface—This is usually one of close proximity, typically less than 50 feet. The transfer rate across the interface is dictated by the speed of the device (data clocking rate). The commands used across the interface are device dependent and both the timing constraints and the individual commands are uniquely device dependent.
2. Control interface—The distance is longer, and is typically in the range of 50 to 200 feet, depending on the size of the configuration. Data transfer rates are a function of where the buffering is located; that is, if buffering is done at the CPU, then the rate is determined by the head of string because it is transferring data at the rate of the device. If buffering is incorporated in the head of string, then speed is a function of how fast the CPU chooses to accept data because speed matching is a func-

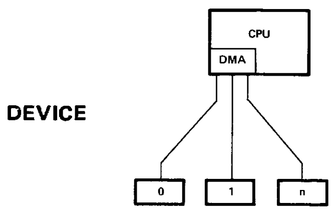


Figure 1—Typical SMD application

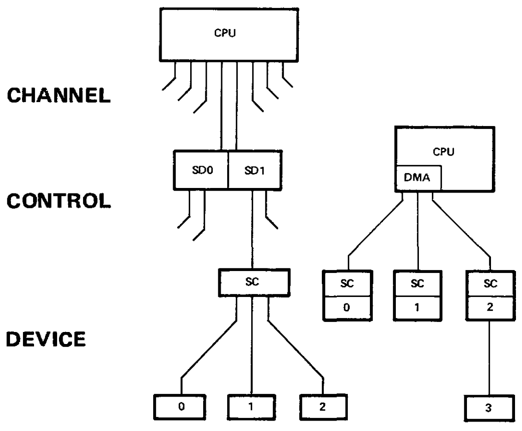


Figure 2—Interface environments

tion of buffer management. It is desirable that commands be class specific (disk or tape) rather than device specific (type of disk or tape).

3. Channel interface—Distances are expected to be much longer, in the range of 200 to 400 feet or more. CPUs that use channels are fast enough that performance is either controller- or interface-limited. Commands are disconnected so that controllers may be multiplexed; it is desirable that they be function generic (i.e. random/sequential, read/write) to mask all device uniqueness.

In the past, as the functionality of an interface increased so too did its manufacturing cost. It was not practical to limit the number of different implementations, because the costs associated with each tended to vary dramatically. To the degree to which each interface varies, there are unique costs incurred, and the more there are the higher the levels of support in engineering, training, and spares that are required.

Today, more cost is associated with the development and maintenance of system components than with manufacturing cost. Life-cycle cost analysis proves that it is cheaper to use common parts than it is to optimize individual items (there are huge savings in spare parts alone). Interfaces have to become a generic system component in order to achieve similar life-cycle cost savings.

The IPI is a modular interface that is layered in an architecture similar to that of the Open Systems Interconnect (OSI) model. It is not a proprietary interface; instead, it represents a combination of the best features of all the interfaces that were proposed to the committee.

Since terminology does not have the same meaning to everyone, the proposed standard has defined several terms to avoid confusion; for example, in an intelligent environment the term *host* is used to identify the master and the term *unit* is used to define the slave (which controls devices). In a device-oriented environment the term *unit* defines the master and the device is the slave. Both the physical interface[1] and the logical interface[2] definitions of the IPI contain a glossary of terms and definitions.

It is practical to use the IPI as a channel, control, or device interface, depending on the repertoire of commands used. The master-slave configuration is used to obtain simplicity and performance, and there are two major divisions: The physical interface defines the interconnection for transferring information; the logical interface discriminates between information as either the operations to be executed or the data to be transferred.

The two divisions operate independently of each other, in the sense that it is possible to replace one physical interface with another and have the logical interface remain the same (e.g. open-emitter logic using coaxial cable for long distances may be replaced with a fiber optics loop). Alternatively, it is possible to retain the same physical interface and replace or extend the logical interface command repertoire to provide a database backend processor in place of a disk controller.

The physical interface defines the mechanical, electrical, and protocol specifications necessary for transferring information across the IPI. The multiple interconnect specifications listed in Table I are provided so that cost considerations of in-cabinet harnesses versus external cabling can be managed by the manufacturer.

Table I—Interconnect classes

	DISTANCE	ELECTRONICS	MECHANICAL
A	3 METERS	THREE STATE	RIBBON CABLE TO HEADER
B	15 METERS	OPEN COLLECTOR	
C	65 METERS	DIFFERENTIAL	
D			CABLE TO BULKHEAD
E	5 METERS	OPEN EMITTER	RIBBON CABLE TO HEADER
F	125 METERS		COAX TO BULKHEAD

Different applications need different physical interconnections, thus the choice between logic families and cable configurations. Class A is suited to small systems and uses the same logic, connectors, and cables as defined in the rigid disk interface (BSR X3.101M). These limit cable lengths to 3 meters, are cheap, and are quite suitable for in-cabinet configurations. At the other extreme is the requirement for large mainframe complexes that place subsystems hundreds of feet away from the central-processor complex. Industry has a diverse range of requirements, so a standard must not limit the environments where it can be used.

Intermixing of the different classes of physical interconnects is likely to occur often. Classes C and D have the same electrical specifications but use different cables and connectors—for mainframe installations Class D is suitable for the long cable lengths between system controller and peripheral-subsystem strings, but within a string Class C would be used to save cost. As another application, a high-capacity disk subsystem using Class D to the CPU could incorporate a small cartridge tape in each cabinet to serve as an audit trail by logging activity against the disk. The interconnect to the tape cartridge would be internal to the disk cabinet and able to use Class A.

As seen in Figure 3, 5 signals (three out and two in) are used to control the interface. One signal change at a time is alter-

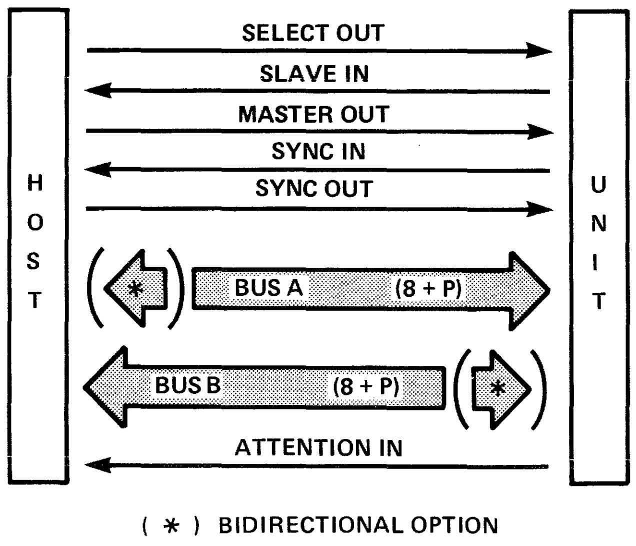


Figure 3—Signal definitions

nated between master and slave to provide complete handshake control of the interface. There are two unidirectional 8-bit buses with a transfer rate that exceeds 3MBs, and as a performance option the two can be used bidirectionally at over 6MBs. The remaining signal is used as an ORed attention line to advise the master of interrupt conditions in the slaves.

Implementing the IPI requires but a simple state machine, virtually free of timing dependencies. With 5 control signals there are only 32 possible states. Of these, 12 are illegal and 6 are resets. That leaves only 14 states needed for normal operation, and the 9 sequences require fewer than 30 transitions between states.

The few signals, small number of states, and the limited sequences reduce the size and complexity of logic necessary to control the interface. Everything can be managed by a micro except recognition of reset, a precaution necessary to recover from microprocessor failure, microcode loops, bugs, and so on.

Implementation techniques vary widely; one uses a Z8 micro and a handful of chips, one is using custom gate arrays, and another uses the Signetics 50ns 82S105 FPLS as a "state machine in a chip." Common to all is that they are cheap by comparison with the interfaces they are replacing.

A microprocessor is not expected to handle the transfer function, unless the rate involved is quite low. The slave defines the transfer clock rate so that it can occur at the maximum rate speed of the device (or buffer, if any). Transfer rate is a product of cable length, skew, and technique used. If transfers between the master and the slave are interlocked, all information is "handshaked" over the bus and there are delays (in both directions) that are associated with cable length.

Data streaming may be incorporated in the intelligent host-unit environment to double the transfer rate by not interlocking the syncs (data clocks). Information on the buses is strobed by a sync (clock pulse), and there is no interlocked handshake to confirm receipt of the pulse. This is similar to the way most bit-serial device interfaces clock data into the receiver. The effect of cable delay is that the interface contains multiple clock and information pulses at the same time. The host responds to each SYNC IN with a corresponding SYNC OUT, so that when a transfer is terminated the response syncs are still "on the cable" and it is the responsibility of the unit to check that there was the same number of syncs in and out. The way in which data streaming has been implemented in the IPI permits a host to operate transparently between both interlocked and data streaming modes. If the width of the SYNC IN pulse is greater than the cable delay, the transfer is interlocked.

An important feature is the ability to intermix devices with widely varying transfer rates on the same cable. Transfers can be made in any combination of interlocked or data streaming, 8 bits or 16 bits wide. The capacity of up to eight units, each with up to 16 devices, gives a maximum of 128 devices addressable by the host on a single daisy-chained cable.

As illustrated in Figure 4, functions on the IPI are complementary to each other. At the physical interface a bus exchange is used to frame an information transfer, which can be data, an operation command, or an operation response. The bus control defines the parameters (operation or data, in or out, 8 or 16 bits wide) of the information transfer, and slave status reports on its completion (success or failure).

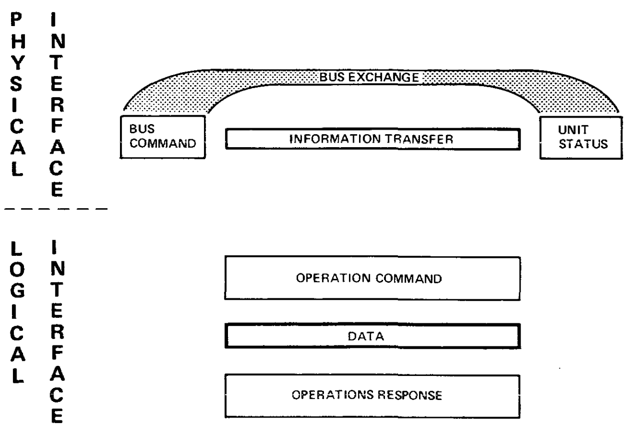


Figure 4—Interface structure

The contents of an information transfer are transparent to the physical interface, which is simply a vehicle for using the IPI as a device, control, or channel interface. The logical interface frames data transfers with an operation command and its associated response. The "intelligence" of an IPI implementation is a product of the command repertoire defined between master and slave. The repertoire itself is a product of the environment in which the IPI is used:

Master-controlled devices.—The master functions in a fashion similar to that used with device interfaces. This type of operation is presumed to offer little or no increase in functionality over rigid disk interface (RDI) or storage module drive (SMD) types of interfaces. Vendors that have devices installed and operational with current software need to install enhanced peripherals. It is much easier to introduce new hardware into a system when there is no need for new software. Software changes can be introduced gradually if the new hardware incorporates new functionality.

It may be impossible to install new hardware on the old interface; for example, disks presently available with a 3MBs data rate (24 MHz) cannot be implemented with an SMD interface. The IPI can be used to emulate the SMD command set to the master so that no software need be changed. The extended functionality offered by the IPI can be taken advantage of in stages. For example, to improve serviceability the first stage may be to provide new maintenance and diagnostics capabilities. Operating system software changes usually take longer to implement, but as they occur they can be made without hardware impact.

Some uses of the IPI will be device and vendor specific if it is necessary to extend the command repertoire beyond that defined in the logical interface. The proposed standard does provide for this kind of use as a vendor-unique application, because such implementations provide the vendor with a migration path to compliant logical interface operation.

One example of this kind of use is defined as synchronous because it represents a time-critical operation executing between CPU and device. This situation can occur when the IPI is used subservient to another interface such as the FIPS 60

Block Multiplexer Channel. The channel issues channel command words (CCWs) that are chained together and require responses within a very specific time period. On disks with count-key-data (CKD) records the channel must actually "turn-around" during the gap times of the disk. If the CPU-SCU and SCU-disk interfaces are FIPS 60 and IPI respectively, the IPI must run synchronous to the timing constraints of the channel. Similar situations may occur whenever the IPI is dropped into any pre-existing environment.

Shared control of devices.—This is an intelligent environment where host-unit operation may have any of a wide range of characteristics—from one where the host dictates control of the devices through the unit, to a more independent mode of operation where the unit controls the devices. The command repertoire is oriented to device characteristics (disk, tape, etc.) and uses device-generic commands that support logical addressing (e.g., relative block) and a format (e.g., fixed sectors). This level of operation permits the unit to incorporate such features as buffers, error correction, error retry, and defect handling.

The command repertoire defined is extensive, because it incorporates pathways to let vendors migrate to more intelligent use of peripherals. The commands defined not only replicate those to be found in the SCSI (X3T9.2/SASI) and ISI (CDC/MPI), but represent a superset that permits application over a much more diverse and varied set of configurations and architectures. Since the objective is to permit the vendor to coexist with the installed environment there are no predefined characteristics that require buffering, error correction, error recovery, and so on in the unit—if they are present, the command repertoire can use them.

Different performance profiles can be configured—if commands are issued as individuals, only one operation per device is active at a time, and it has no timing-critical dependencies. If performance is a criterion, the unit may operate with queued commands. In this implementation the unit is capable of stacking multiple operations per device on behalf of the host, and of servicing them in a manner defined by host and/or unit algorithms. One example of this would be performance improvements on disks, obtained by the unit queueing requests and executing them in a sequence according to some seek-optimizing algorithm (unless overridden by a host-defined priority request).

Unit-controlled devices.—Operations are host-unit and require that a number of architectural attributes such as memory management reside in the units. The control of devices is distributed, since their management is within the scope of the unit; the host has limited control, if any, over the details of device manipulation. Either individual or queued operations are executed, depending on the capability of the unit.

At this level of operation few commands are needed, because each one is powerful and rich in functionality. The host is cognizant only of the attributes of the data (i.e. random, sequential, input, output; thus there are no device-dependent characteristics. The addressing structure does not recognize physical boundaries, since data is allocated in files on logical volumes, which may or may not be on a single physical device.

When execution occurs on the IPI for this type of implementation it is probably being used as a channel. The level of capability necessary in the unit represents a high level of functionality that is relatively high in cost. As such, it is assuming a considerable degree of the data manipulation burden of the host.

To support these variations in environment and application, the IPI has applied techniques implemented successfully in communications networks to support multiple environments of use. The logical interface uses message packets for operation commands and operation responses, with sequences to perform a task constructed from a series of macros. For example to read data in from a slave to the master requires a sequence of five macros:

SELECT
OPERATION COMMAND
READ DATA
OPERATION RESPONSE
DESELECT

Only simple operations are defined for device interface applications, but in the intelligent host-unit environments the complexity of the tasks that can be accomplished is limited only by the functionality incorporated in the unit. The operation commands and operation responses are message packets of variable length that are transferred across the physical interface as information. (See Figure 4.) A command packet has a generic root of 14 bytes plus, if needed, a variable-length parameter list. A response packet images back the generic root plus 4 bytes of status and, if needed, a variable-length parameter list. The commands are broken up into the following groups:

READ LOGICAL	COMBINATION READ/WRITE
READ PHYSICAL	CONTROL
WRITE LOGICAL	DIAGNOSTICS
WRITE PHYSICAL	OTHER

Some of the commands are generic to all uses (e.g. some control commands), but others are very specific (e.g. a physical read to a disk). The read and write physical commands are unique to the device interface level of operation. In most applications only a subset of the available command repertoire is needed.

It is unnecessary to define the ultimate environment for use of the IPI in advance. Since commands are assumed to be implemented in microcode, unique requirements can be added to the chosen repertoire to meet specific vendor needs and problems. Over time, as systems adapt to higher levels of function the unit will grow in complexity as it incorporates functions currently considered part of the operating system responsibility. The most fascinating part of such a scenario is that the same physical interface can support all of the envisaged environments, and the IPI can evolve to meet the vendor needs.

To assist in making this kind of growth feasible and modifiable, the IPI provides extensive housekeeping capabilities that permit the master to configure the slave to the required environment of use. Each slave has attributes that can be examined by the master—some, such as physical record size, will be soft on one device but hard on another (e.g. on imbedded servo disks the physical record size cannot be altered). The combination of attributes offered by a slave defines the ways by which the master can customize it to the particular installation. Soft attributes can be selected and modified by the master to tailor the slave, while hard ones cannot be modified and must be accepted as given. Attributes will normally be established after initial power-on, but they can be used dynamically to reconfigure systems to adapt them to changing workloads or cover hardware outages.

Using the IPI as a device interface has been discussed already; it can represent major cost savings. Tapes are parallel, and all the new coding schemes (2/7, 3PM) for disk depend on byte constructs (data is recorded bit-serially, and the data separator does the necessary conversions). As shown in Figure 5, a device interface such as the SMD requires that it be converted back to bit serial to go to the outside world—where it is reconverted into bytes again by a head of string or a DMA controller. Given the increasing use being made of high-density coding schemes this makes no economic sense at all—the cheapest, and also the fastest, approach is to leave the data parallel from the data separator on, as can be done with the IPI.

The on-board electronics are not the only savings—there are even more significant economies in cabling costs. Bit-serial device interfaces are radial (Figure 1), and usually have two cables (the SMD has one for control and one for data). A parallel device interface requires only a single daisy-chained cable (Figure 7), and can operate at much higher data rates. When the cost of the separate cables and the difficulties of installation are addressed, the parallel IPI provides savings to both the manufacturer and the installer.

At the other end of the spectrum, the IPI is very suitable for use as a channel, because it provides the ability to operate under the FIPS 60-63 architectural constraints. A channel interface has to be able to interconnect over long distances, at a very high data rate. Cabling costs are very high, electromagnetic interference (EMI) is a serious concern, and grounding isolation is difficult to achieve. The latter concerns make fiber optics appear very attractive, but a fiber optics implementation of FIPS 60 requires three fibers and cannot be installed cost-effectively.

A single fiber loop or ring can be cost competitive with the current FIPS 60 type of coaxial-cable connections. The IPI has been designed with this consideration in mind, and a conversion of the physical interface such as that shown in Figure 6 has no impact upon the logical interface command repertoire implementation.

One of the biggest problems with EMI compliance is that even though cabinets may individually comply, when they are configured into a system the system may not comply. The cost of testing for compliance is a significant factor that all vendors of computing systems have to face as an ever-increasing burden. When fiber optics is used as an interface there are no intercabinet EMI considerations, which means that system compliance is a near-automatic follow-on after cabinet compliance has been achieved. The FCC and VDE regulations have caused considerable trauma in recent years, and unless there is a switch to fiber, future systems will represent an even worse problem.

The choice of how to implement an IPI is a function of cost and performance—on a small system with limited performance requirements, integrating both functions into the on-board processor makes the most sense. However, for higher-performance systems it is better to have a head of string on a separate board that uses a fast processor. The advantage to this structure is that the systems integrator only has to support one interface, and it is easy to offer simple performance upgrades from device-only configurations by adding heads of string boards.

One point that must not be overlooked in implementing the IPI is *undersizing* the microprocessor. The physical interface can be controlled by an inexpensive slow processor, but as the level of intelligence offered increases, so too does the need for faster processing. An IPI used to interface devices at a primitive command repertoire level does little processing, but as functionality is added the power to process commands at the logical interface must be provided—if it is not, the unit will take too long to respond to commands and will be limited in performance. This may be acceptable in some low-end appli-

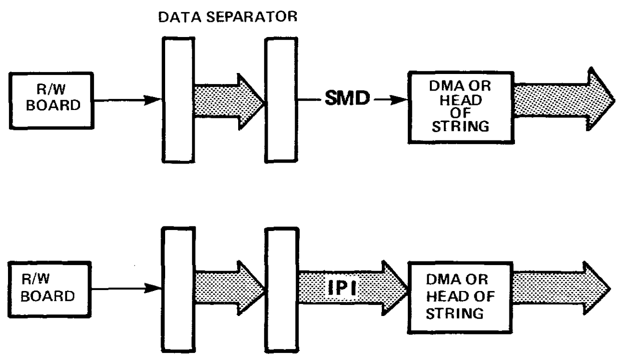


Figure 5—Device interface comparison

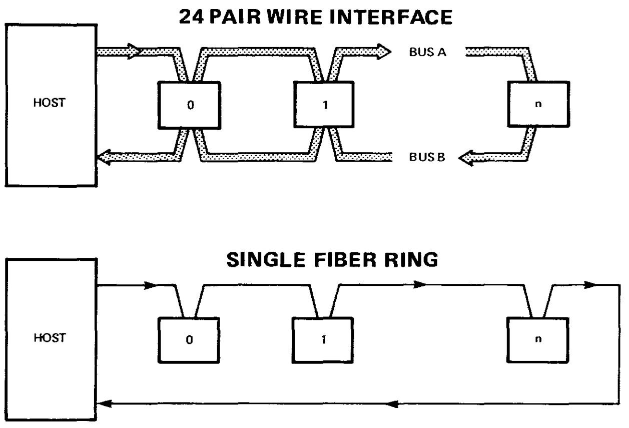


Figure 6—Parallel wire to fiber loop

cations, but where intelligent processing is needed, either a fast processor in every device or a fast processor at head of string is needed.

Figure 7 illustrates how a mainframe can configure the IPI as all three interfaces (channel, control, and device), a large mini can configure it as two (control and device), and a small mini can configure it as one (device). The configuration is made up as follows:

1. The mainframe CPU is connected by a channel interface to an SD controller.
2. The SD and a mini, via its DMA controller, are daisychained to two dual-port heads of string (one standalone and one integrated unit/device combination).
3. The standalone head of string and a small mini, via its DMA controller are daisychained to dual-port devices.

In the control or intelligent interface environment the X3T9.2 project for the SCSI can be considered an alternative to the IPI. The SCSI has an 8-bit bus and offers a peer-to-peer capability, such that units attached to it may assume the role of either initiatior or target. There is an arbitration algorithm that settles possible conflicts in use on interfaces less than 30 meters in length (which is adequate for small systems).

There are implementations of the SCSI available in custom LSI, which makes the job of systems integration much easier

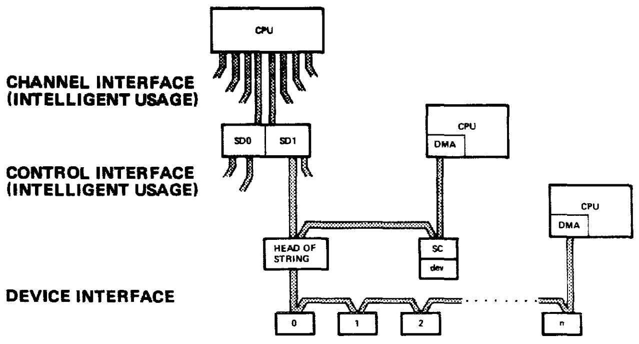

CPU
CHANNEL INTERFACE (INTELLIGENT USAGE)
SD0
SD1
CPU
DMA
CONTROL INTERFACE (INTELLIGENT USAGE)
CPU
DMA
HEAD OF STRING
SC
dev
DEVICE INTERFACE
0
1
2
n

Figure 7—Configuration flexibility

for both the device and the controller manufacturers. Not only is the physical interface integrated but so too is the basic command repertoire. This makes for a high degree of applicability and compatibility in future SCSI installations.

The SCSI can coexist with the IPI because both use a structured, modular approach to implementation and are oriented to use in new systems applications where there is no need for compatibility with existing software and hardware. Equivalence between SCSI and IPI configurations can be obtained by using an SCSI-like logical interface command repertoire. An SCSI complex could be configured with one node dedicated to high performance disk activity that uses IPI interfaces to the disks. The operating system software in such a complex would be almost totally unaware that it was operating with two different physical interfaces except at the I/O dispatcher level.

When a vendor has to configure systems where multiple levels of interface must be supported, the IPI is unique in its portability of function. It is possible to use the same physical interface as device, control, and channel with the only difference being the command repertoires at each. Even more likely is that the same physical interface will be used with more than one level of logical interface capability. As new peripherals and software are introduced, the command repertoire can expand and grow to suit higher levels of functionality. In this way the IPI serves as a migration and growth vehicle for vendors.

There is no equivalent interface offered by any vendor that offers quite the same breadth of application. The IPI represents the essence of simplicity, and very little has been compromised for its application in any of the three interface definitions. This is a testimony to the efforts and capabilities of the ANSC X3T9.3 membership, who have given generously of their time and effort. The IPI is not, and never was, a proprietary interface—and this may in itself be the major reason for its universality, simplicity, performance, and ease of use.

REFERENCES

1. ANSC Document Number X3T9.3/176
2. ANSC Document Number X3T9.3/82-19

Progress on the network layer of the OSI reference model

by PETER F. LININGTON
Rutherford Appleton Laboratory
Chilton, Didcot, United Kingdom

ABSTRACT

Work in the International Standards Organization on communication protocols is structured in terms of the Open Systems Interconnection Reference Model. The network layer of this model provides independence of network technology, including details of routing and switching. The standardization activities involve definition of the service provided by this layer in the Open Systems Interconnection structure, the organization of functions within the layer, and the specification of protocols to support them. The structure described allows communication systems to exploit a wide range of different network types while preserving a uniform set of user facilities. The work on the network layer is of vital importance to the acceptance of the new communication standards and their early application to practical networking problems.

A. THE OSI ARCHITECTURE

Work on a general set of standards covering many aspects of data communication has been in progress within the International Standards Organization (ISO) for several years. The project is known as Open System Interconnection (OSI). An OSI reference model[1] has been created to provide a structure for work on the protocol standards needed for interconnection of computers. The aim of this model is to encourage parallel work on different facets of the interconnection problem by placing them within a consistent architectural framework. This can be achieved by dividing the functions that need to be performed into a number of nested layers. Each layer is based on the capabilities resulting from the more basic functions in the layers below, which are said to provide a supporting service. Each layer in turn offers some more powerful set of capabilities to higher layers, which communicate by using them; these capabilities are represented as the service provided by the layer.

The functions allocated to the network layer in the reference model are primarily those involved in management of location. It is concerned with routing and switching mechanisms, and with the combination of different communication technologies. In this it can be contrasted with the layer above, the transport layer, which is independent of these considerations and provides a quality of service enhancement resulting from protocol exchanges between the single pair of entities that use the network service directly.

The basic reference model produced by ISO was concerned with communication describable by the operation of a connection. It identifies connection establishment, connection use, and connection termination. However, further work is now in progress to extend the Reference Model to include operation that does not require a connection; this so-called connectionless operation can be used to model certain existing types of networks; it has thus become of considerable practical importance in discussions of the network layer.

B. TECHNICAL OBJECTIVES OF NETWORK STANDARDIZATION

The standardization committee concerned with the network layer within ISO is Subcommittee 6 (data communication) of Technical Committee 97 (data processing). This committee cooperates closely with Subcommittee 16, which is concerned with the higher layers of the reference model, and with general architectural questions. SC6 brings together many different networking interests. There are representatives of computer manufacturers, common carriers, and computer users. They reflect interests in both local and wide-area networks, operating both in the private and in the public domain. The discussions therefore range over a wide spectrum of existing and projected network types. The unifying factor in the discussions, however, is a desire for widespread simple interoperation. The aim of OSI in general is to remove the technical impediment of unnecessary variety from the communication process. In the network layer, the emphasis is on allowing equipment attached to any type of network to communicate, via suitable intermediaries, with equipment attached to any other type of network. Particular importance is given to the interconnection of private local area networks via wide-area public networks (see Figure 1). The main thrust of the work is therefore toward unification of the user view of many different technologies.

C. TECHNOLOGICAL VARIETY

The main barrier to the simple and uniform view of communication desired is the wide range of technological solutions

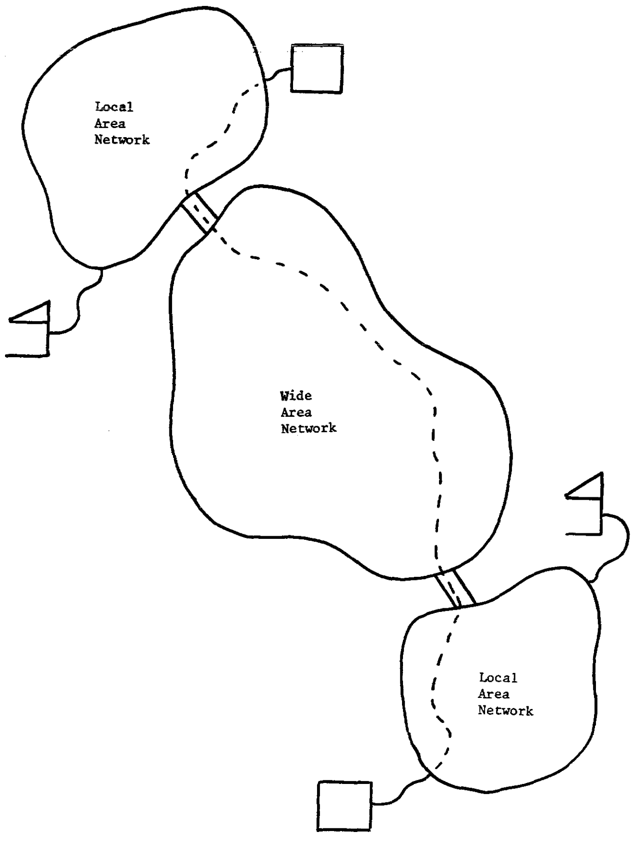


Figure 1—Tandem local-area and wide-area networks

available to network constructors. Major differences that need to be resolved are

1. The difference of approach between circuit-switched and packet-switched communication. This difference in the way the communication resource is managed has wide-ranging consequences in terms of the richness of the service the user sees. As a consequence of the resource-sharing and queueing facilities included within packet-switched networks, there are user-visible control functions such as reset and interrupt mechanisms. These features do not form part of the network when a channel is dedicated to the user in a circuit-switched system.
2. The difference between connection-oriented and datagram-oriented networks. Connections provide a certain level of communication management that must be provided outside the network in the datagram case. However, the provision of connections is not without cost, and there is as yet no general agreement on a single optimum solution.
3. The variation in costing of communication. The major difference here is between private networks, charged on the basis of capital depreciation, independent of detailed use patterns, and public networks, charged on the basis of actual use. Within the public domain, different tariff structures give radically different weights to data-transmission, connection-establishment and connection-holding time. These differences all tend to be reflected in different design choices for the network protocols.

D. ADMINISTRATIVE REGIMES AND NETWORKS

One of the major problems in attempting to establish a uniform communication system is the fact that different components are likely to be managed by different organizations. The resultant division of responsibility implies a need for a careful definition of what a network is. This is not a simple matter, because the standardization of protocols turns on who caused each message to be sent and who took what action on it, not on physical groupings of equipment. For example, a PTT-operated network contains many functions. Some of them are concerned with allowing user-to-user communication; some of them are involved with value-added user-to-network communication. Conversely, some of the basic communication functions are in equipment supplied by the network user.

In consequence of this distribution of function, the term *subnetwork* has been introduced to describe the physical collection of equipment, in contrast with an *OSI-Network,* which bounds a particular collection of functions. A subnetwork may provide either more or less function than the idealized OSI-Network. Any function that is not provided must be made good by the users of the subnetwork, in cooperation with one another. This division of function is reflected in the organization of networking standards, described in section F below.

In addition to straightforward protocol implications, the division of responsibility for communication has some more subtle implications with regard, for example, to addressing or network maintenance.

The problem of addressing is that the organizations responsible for the different elements will need to act independently in allocating addresses. It would not be acceptable, for instance, if a private network manager had to liaise with the PTT providing him or her with remote access whenever he or she wished to allocate a new address. Where the organization is hierarchical, this problem can be mitigated by the use of hierarchical addressing schemes, although problems of the unpredictable size of the address space still remain. When the organizations are autonomous peers, however, there are more severe problems, which now seem soluble only by construction of an artificial hierarchy. Other problems arise when an organization divides, when two previously independent organizations are combined, or when existing functions are relocated at a different site.

In considering network maintenance, the major issue is the allocation of responsibility for errors. The user of the network will receive error indications when unrecoverable errors occur, and may need to know which component has failed in order to apply pressure on an organization that is not providing him or her with the contracted quality of service. These issues of diagnosis raise management problems that have not yet been resolved in the standards discussions, but that do have an impact on the design of network protocols, because additional information must be passed with error reports where more than two components are involved.

E. THE NETWORK SERVICE

The general architectural approach of defining abstract services before fixing protocol detail was introduced above. The development of a network service definition is well advanced.[2] It is proceeding by collaboration between ISO and CCITT. The two organizations have been holding meetings alternately, taking note of each other's progress, so that the technical content of their two service descriptions is well coordinated, and the drafts are fairly stable. The major area of instability at present centers on the need for and precise definition of an expedited-data (interrupt) facility.

The network service provides for the transparent exchange of network service data units (NSDUs) between transport entities. Transport entities are unambiguously identified to other transport entities and to the network service provider by their network addresses.

The network service provides to the transport entities independence from routing and relaying considerations. This includes the case where several transmission resources are used in tandem or in parallel. It makes invisible to transport entities how the network layer uses underlying resources such as data-link connections to provide the network service.

The network service defines

1. A connection that may be established or terminated between the network service users for the purpose of exchanging data. More than one network connection may exist between the same pair of network addresses.

2. Associated with each connection, certain measures of quality that are agreed on by the network service provider and the network service users when the connection is established.
3. Means of transferring NSDUs on a connection; the transfer is transparent, in that the boundaries of NSDUs, the sequence of the NSDUs, and the contents of NSDUs are preserved unchanged by the service, and in that there are no constraints on the data values imposed by the service; the transmission of these data is subject to flow control.
4. Means by which the connection can be returned to a defined state, and the activities of the two users synchronized by use of a reset service.
5. Means for the service user to confirm receipt of data.
6. Means for the service user to send expedited data that are not subject to the normal flow control.
7. The unconditional and therefore possibly destructive termination of a network connection.

The description of the components of the network service is in terms of exchange of primitive actions, or primitives for short, between the users and the provider of the service. For example, the significant events in the establishment of a connection are described as follows:

1. The calling user issues an N-CONNECT request primitive to the service provider.
2. After a certain time, the service provider issues an N-CONNECT indication to the other service user.
3. If prepared to accept the connection, the called user issues an N-CONNECT response to the service provider.
4. Finally, after some time, the service provider issues an N-CONNECT confirm to the calling user, showing that the connection is complete.

The groups of primitives defined in the current network service definition are given in summary form in Figure 2. The use of primitives allows the simple expression of the constraints on the available sequence of actions. It does not, however, allow the direct expression of more complex communication properties, such as the existence of flow control or the properties of expedited data. To this end, the operation of the service provider is modeled in more detail by the operation of a dynamically modified queue.

This description applies to the connection-oriented service. The description of the connectionless service[3] is not so well advanced.

However, the service is inherently simpler, in that it allocates more of the functions to the communication user. The service description can therefore be expected to progress rapidly.

Nevertheless, there are problems in the description of the properties of real networks that require the interrelation of different actions made by the service users. For example, most of the current local-area technologies can be described by a connectionless service but also have the property of maintaining the sequence of user actions. This sequencing property is of importance when one is considering user-to-user protocols, and queue-like models for expressing these properties are being studied.

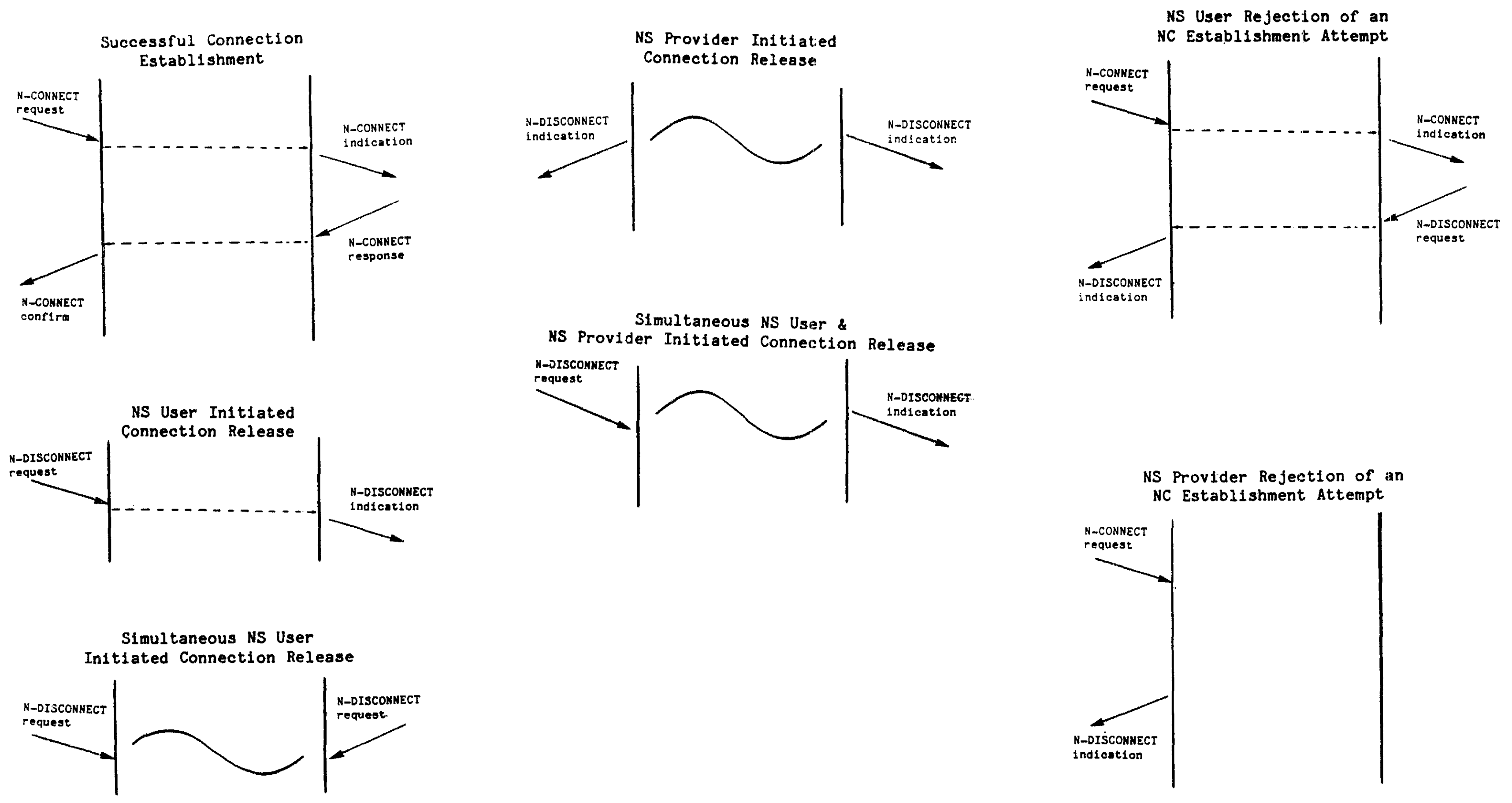
F. INTERNAL ORGANIZATION OF THE NETWORK LAYER

Work has been taking place to define an internal organization for the network layer,[4] in order to allow the interworking between different subnetwork types. The role of the subnetwork in representing the real-world networks, rather than the regimes using particular protocols, was explained in Section D. The result of this analysis of the practical constraints has been the identification of four groupings of functions.

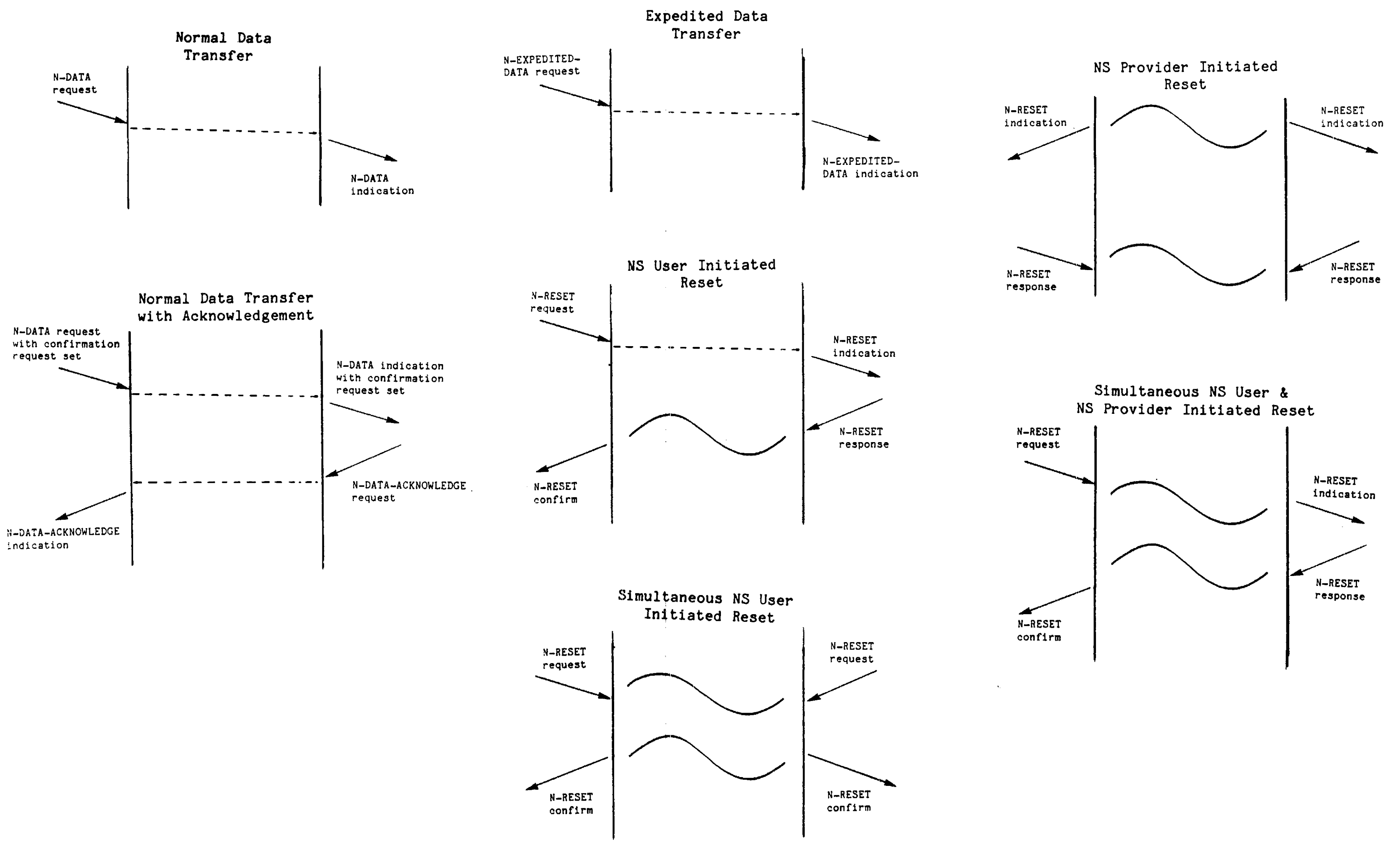
1. The subnetwork access functions, which are associated with those protocols needed to support the direct interactions between a pair of entities using a particular subnetwork type. The operation of these functions can be described abstractly by a subnetwork service specific to the subnetwork concerned. For example, there is a service corresponding to capabilities of an X.25 network, abstractable from (and more stable under review than) the specific X.25 protocol.
2. The subnetwork-dependent convergence functions, which are the functions that, for a particular subnetwork type, are not included in the set of subnetwork access functions but are needed to convey the information required to support the OSI network service across the particular type of subnetwork.
3. The subnetwork-independent convergence functions, which are the functions that are needed to convey the information required to support the OSI network service but can be defined without reference to a particular subnetwork type.
4. The concatenation and routing functions, which are the functions needed, in addition to the subnetwork access and subnetwork-dependent and independent convergence functions, to concatenate a pair of subnetworks so as to provide the appearance of a single, uniform subnetwork. These functions correlate the activities related to the individual subnetworks and can be defined in terms of the services of the two subnetworks. They are localized functions and do not add to the protocol required.

The division of function is shown diagrammatically in Figure 3.

The distinction between subnetwork-dependent and subnetwork-independent convergence functions needs some further explanation. It arises from the wide range of technical and economic constraints applied to the network layer. An apparently simple strategy for convergence would be to set a minimum functional requirement for any subnetwork, and place almost all the functions in one standard, universal network protocol. The subnetwork-specific work to be done would then be the definition of an essentially trivial mapping of the minimum requirements onto the particular subnetwork


Successful Connection Establishment
N-CONNECT request
N-CONNECT indication
N-CONNECT response
N-CONNECT confirm
NS User Initiated Connection Release
N-DISCONNECT request
N-DISCONNECT indication
Simultaneous NS User Initiated Connection Release
N-DISCONNECT request
N-DISCONNECT request
NS Provider Initiated Connection Release
N-DISCONNECT indication
N-DISCONNECT indication
Simultaneous NS User & NS Provider Initiated Connection Release
N-DISCONNECT request
N-DISCONNECT indication
NS User Rejection of an NC Establishment Attempt
N-CONNECT request
N-CONNECT indication
N-DISCONNECT request
N-DISCONNECT indication
NS Provider Rejection of an NC Establishment Attempt
N-CONNECT request
N-DISCONNECT indication

(a)



(b)

Figure 2—Summary of network service primitive time sequence diagrams

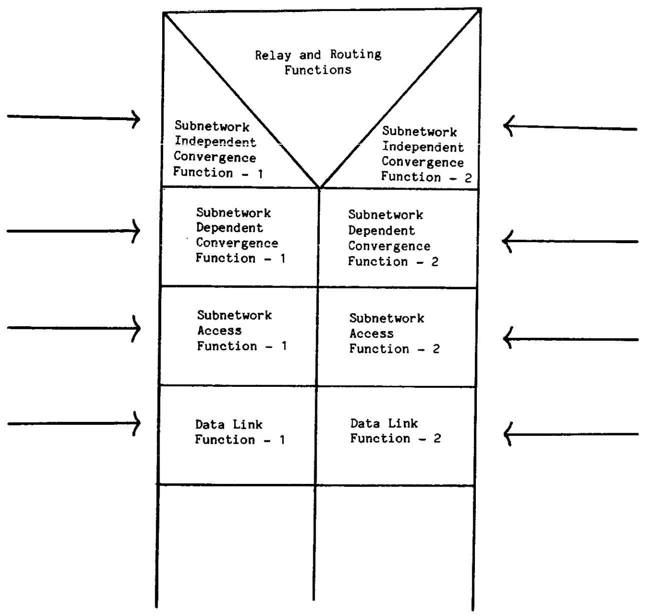

Relay and Routing Functions
Subnetwork Independent Convergence Function - 1
Subnetwork Independent Convergence Function - 2
Subnetwork Dependent Convergence Function - 1
Subnetwork Dependent Convergence Function - 2
Subnetwork Access Function - 1
Subnetwork Access Function - 2
Data Link Function - 1
Data Link Function - 2

Figure 3—Internal organization of the network layer

service. This is the philosophy adopted by the designers of so-called internet-protocols. An alternative strategy, the so-called hop-by-hop enhancement approach, involves identifying as many parallels as possible between the subnetwork and the idealized OSI network service, and applying the minimum enhancement necessary for each subnetwork. Broadly speaking, the first approach minimizes implementation cost, while the second minimizes operational cost. In consequence of the wide range of interests to be satisfied, neither of these extreme approaches is altogether satisfactory. The division of functions into subnetwork dependent and subnetwork independent is an attempt to provide a framework allowing use of the best features of the two approaches on a case-by-case basis. It emphasizes common elements, while allowing use to be made of the strengths of the individual subnetworks when this gives rise to economic benefits.

In the long term, the existence of an agreed ISO network service will affect the activities of subnetwork providers. The evolution of X.25 and of the future ISDN services may well bring them closer to the OSI network service, so that both the convergence functions become null. On the other hand, the work on local-area network standardization is at present essentially unconstrained, and the subnetwork access function and the subnetwork-dependent convergence function are expected to be null, leaving the subnetwork-independent, internet-like protocol using a data-link-like service.

This internal organization of the network layer is still the subject of active debate, but it does seem to offer a way for progress to be made in a very heavily constrained field.

G. FUTURE WORK IN ISO

The connection-oriented and connectionless network service definitions can be expected to stabilize in the near future, and this stability will be reflected by their progress from the technical to the procedural phase of standardization, starting with their formal registration as ISO Draft Proposals.

The major activity that is only just beginning is the specification of the protocols which support the convergence functions. Distinct subnetwork-dependent convergence protocols will be needed for the major technologies. Initial work is expected to cover at least the local-area network field and the widely available PTT networks based on recommendations X.25 and X.21. The emerging ISDN standards will also need to be covered. Moreover, it is in the nature of the variety reduction that lies behind the work that it will continue for as long as new types of networks continue to develop and so need to be assimilated. The benefit of standardization will, however, be the ability for equipment from diverse suppliers to communicate easily and efficiently whenever desired.

REFERENCES

1. *Information Processing Systems—Open Systems Interconnection—Basic Reference Model.* ISO DIS7498.
2. *Open Systems Interconnection—Network Service Definition.* (September/October 1982) ISO/TC97/SC6 N2610.
3. "Working Draft for an Addendum to the Network Service Definition covering Connectionless Data Transmission." ISO/TC97/SC6 N2611.
4. *Internal Organization of the Network Layer.* ISO/TC97/SC6 N2613.

The technology of digital speech: compression, editing, and storage

by R. E. CROCHIERE and J. L. FLANAGAN
Bell Laboratories
Murray Hill, New Jersey

ABSTRACT

The advantages of representing speech in digital form have long been recognized in digital transmission and switching applications in telephony. This digital speech technology is now rapidly expanding into new and diverse areas of application in speech communications and man-machine interaction. It is being spurred by new advances in the fields of digital communications, signal processing, computing, and very-large-scale-integration (VLSI) device technology. In this paper we briefly outline some of the key elements in this technology and focus, in particular, on current research and potential application of speech compression, editing, and storage.

SPEECH CODING

Intense research activity on speech coding and compression over the past three decades has produced numerous algorithms and techniques.[1] Typically these methods trade on three basic factors in coder design: required bit rate, algorithm complexity (cost of implementation), and performance (quality of intelligibility).

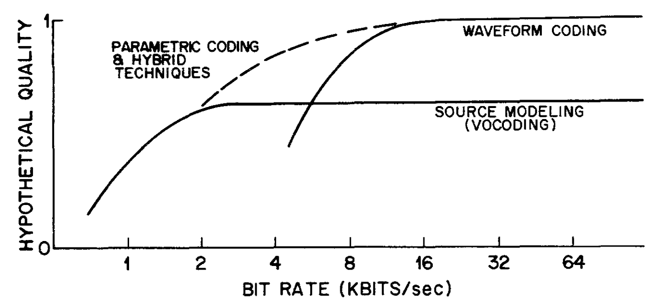


Figure 1—Speech coding and vocoding: Quality as a function of bit rate

Figure 1 summarizes impressionistically the state of the art of speech coding and vocoding in terms of quality as a function of bit rate. The vertical scale represents a simplified hypothetical measure of speech quality where a value of one implies a quality that is essentially indistinguishable from the original input for telephone bandwidth speech, and a value of zero implies a quality that is extremely poor and unintelligible.

SPEECH EDITING AND MANIPULATION

Once in digital form, speech can readily be stored, transmitted, or manipulated. In potential applications such as voice-mail, store-and-forward, or information retrieval, it may be desirable to have additional flexibility to modify and edit speech messages in ways similar to those used today to edit text. Key elements in these types of systems might include algorithms to detect the presence or absence of speech, techniques to insert or delete speech phrases, techniques to modify the duration of silence in the message, and methods for modifying the basic rate of the speech itself.[2]

HARDWARE TECHNOLOGY FOR SPEECH PROCESSING

A major impetus in the evolution of digital speech technology is due to the growing capability of new custom and programmable VLSI devices for real-time digital signal processing. Several of these types of devices have recently been introduced specifically for speech applications. In the area of waveform coding, chips for complete μ-law PCM A/D and D/A conversion (including antialiasing filtering) have recently been developed and are of interest in applications of digital telephony. Chips for ADM (adaptive delta modulation) have also been available and, with growing interest in telephony at 32 kb/s, chips for ADPCM are soon to follow.

In the vocoder area synthesizer chips or chip-sets that realize an electrical model of speech production have recently become available. A notable example that has generated considerable attention is the Texas Instruments speak-and-spell chip, which has been used for voice response in educational toys. A number of other devices have since followed for applications in voice-response and announcement systems.

Another area of VLSI technology that is currently having a strong impact in digital speech is high-speed microprocessors and signal processing integrated circuits. A notable example of this is the Bell Laboratories Digital Signal Processing Integrated Circuit. With this device algorithms such as ADPCM and subband coding have been realized using one DSP for an encoder, a second DSP for a decoder, and a μ-law PCM coder chip for A/D conversion.[3] With five DSPs and a μ-law codec, a 9.6 kb/s speech coder based on harmonic scaling and subband coding has been realized.[3]

For algorithms of high complexity this microprocessor approach is not sufficient. Such algorithms have been realized in real-time, however, using an array processing computer or special purpose hardware.[3] As VLSI technology advances it can be anticipated that this approach will give way to future generations of single-chip DSP-type processors.

REFERENCES

1. Flanagan, J. L., M. R. Schroeder, B. S. Atal, R. E. Crochiere, N. S. Jayant, and J. M. Tribolet. "Speech Coding." *IEEE Transactions on Communications,* COM-27, (1979), pp. 710–737.
2. Flanagan, J. L., J. D. Johnston, and J. W. Upton. "Digital Voice Storage in a Microprocessor." *IEEE Transactions on Communications,* COM-30, (1982) pp. 336–345.
3. Crochiere, R. E., R. V. Cox, and J. D. Johnston. "Real-Time Speech Coding," *IEEE Transactions on Communications* (Special Issue on Bit Rate Reduction), COM-30, (1982), pp. 621–634.

Statistical modeling for automatic speech recognition

by R.L. MERCER
IBM
Yorktown Heights, New York

ABSTRACT

Since 1972, the Speech Recognition Group at IBM's Thomas J. Watson Research Center has worked on a system for the automatic recognition of natural continuous speech. The goal of our research is to produce a voice-activated text-processing system. We have recently worked on a 5,000-word isolated-speech system for recognition of business correspondence and interoffice memoranda.

Although speech recognition is performed with beguiling ease by human beings, the task of transferring this ability to computers has proven quite difficult. Our approach to the problem has been to develop general statistical techniques that allow the recognition system to organize itself when presented with a large sample of speech called training data. In essence, our system refines itself from some crudely specified initial configuration so as to take advantage of regularities present in a speech sample.

Suppose that a talker wishes to convey to a listener the string of words $W1$, $W2, \ldots, Wn$. Each of the words has a pronunciation, in accordance with which the talker will guide his/her speech apparatus through an intricate cascade of gestures, with the result that a series of sound pressure waves will emanate. A microphone placed within the influence of these waves will respond to them in a fashion that is characteristic of the word sequence $W1, W2, \ldots, Wn$.

Let the response of the microphone be denoted by the sequence $A1, \ldots, Am$. We can regard $A1, \ldots, Am$ as a garbled version of the intended message $W1, \ldots, Wn$. The problem of recovering a message from a garbled transmission is the province of communication theory. The problem may be stated as follows: for a garbled transmission $A\ (= A1, \ldots Am)$, determine the message $W\ (= W1, \ldots, Wn)$ that is most probable given A. That is, find W such that $\Pr(W|A)$ is as large as possible.

From Bayes's rule, we have $\Pr(W|A) = \{\Pr(A|W)\Pr(W)\}/\Pr(A)$. Since we are interested in maximizing this for a particular fixed A, we can ignore the denominator on the right. The first term in the numerator constitutes a probabilistic characterization of the transmission channel. It tells us what A's to expect when the talker intends a particular word string W. The second term in the numerator is a probabilistic characterization of the word strings that the talker desires to convey. $\Pr(W)$ embodies all rules of grammar (ungrammatical word strings are less probable than grammatical ones) as well as all semantic constraints (meaningless word strings are less probable than meaningful ones).

It is, of course, impossible to compute these probabilities exactly. We have, however, developed methods for obtaining useful approximations to them. We refer to the approximation of $\Pr(A|W)$ as a statistical model for the speech production process and to the approximation of $\Pr(W)$ as a statistical model for the language. A discussion of these models constitutes the body of the talk.

Implications of VLSI technology for speech processing

by ROBERT W. BRODERSEN
University of California
Berkeley, California

ABSTRACT

In this paper a survey will be made of the state of the art of the application of MOS technology to speech-processing circuits. Projections will be made of the future capabilities of this technology and how it relates to implementation of speech-processing algorithms.

The most significant difference between the design of speech-processing systems of the past and the VLSI systems of the future is that VLSI technology can allow essentially complete integration of the system, including the interface circuits, the signal processor, and maybe even the transducer. This implies a major change in system design methodology since many of the most important advantages of VLSI technology will not be realized if each portion of the system is optimized without consideration of the other components. To cope with this problem, designers have in many cases over-simplified the constraints of the other system aspects in order to focus on their particular interests. Several examples will be given of computational techniques, developed for general-purpose computers, that now may become obsolete because of the new constraints of special-purpose VLSI chips.

Very large-scale integration (VLSI) has come to mean very different things to different people, and a review will be given of some of the interpretations that are presently in vogue. To many (particularly the computer science community), VLSI implies a technology that has the capability to generate chips of such complexity that the designer of VLSI chips is freed from the constraint of minimizing the circuit area, an almost religious preoccupation of the industrial circuit designer. This then allows a variety of new design styles, such as PLAs, gate arrays, and silicon compilers, that have the capability for a high degree of computer automation. On the other hand, the digital signal processing algorithm community views VLSI as the level of integration that will finally allow low-cost implementation of some of their algorithms. VLSI will therefore finally make possible the development of applications of digital signal processing that will have widespread use. To the technologist and industrial I.C. designer, VLSI refers to an integrated circuit that has more transistors than some arbitrary number that ranges from as low as 1,000 logic gates up to circuits that contain more than 100,000 gates. Often a primary reason in the industrial community for determining whether a chip is of VLSI complexity is the aggressiveness of the public-relations and marketing activity associated with the design. An attempt will be made to integrate these various viewpoints to gain a broader perspective on what the actual impact will be of the future developments of the technology.

Network security and vulnerability

by J. MICHAEL NYE
Marketing Consultants International, Inc.
Hagerstown, Maryland

ABSTRACT

Technology-based society demands efficient and reliable communication networks. The introduction of microwave communication links has significantly increased the vulnerability of voice and data communications to electronic interception. With the wide acceptance of automated office systems and the increasing use of communicating word processors and distributed data terminals, proprietary, sensitive, and personal-privacy information can easily be intercepted through passive eavesdropping with relatively inexpensive equipment and limited technical resources. This session will provide an in-depth look at the problem and will review a variety of techniques for protecting facsimile, data, and voice communications.

INTRODUCTION

The age of electronics has provided the ability to intercept voice and data communications for as little as several hundred dollars. In addition, powerful minicomputers, now available at a relatively low cost, can be programmed to generate false data transactions that appear to a legitimate computer system as an authentic data entry terminal. Potential threats for unauthorized interception, or transmission of deceptive messages that appear authentic, can be generated by educated electronics hobbyists or determined adversaries (competitors or foreign governments). Computer crime has been estimated to cost business close to $3 billion annually. The explosive growth of the personal computer market—2,000,000 units now sold, with projections for several million more shipped by 1985—combined with the overabundance of published material, formal training, and self-taught courses on computer technology, have created a new menace: the electronic criminal.

Computer and communications technologies have joined forces to encourage dramatic increases in the volume and speed with which information is collected and distributed. Electronic communication has achieved prominence as the principal method for the distribution of time-perishable information. Geographically dispersed industry and government offices are almost totally dependent on an electronic means of information exchange in order to maintain productivity. Automated office technologies have dramatically increased the number of employees with access to vast computer databases from remote terminals that simply telephone the computer.

Concurrent advances in electronics technology have facilitated electronic surveillance and interception of proprietary or sensitive information. Typically, threats include organized and intentional attempts to obtain economic or proprietary information from the competition; determined attempts to obtain economic and other sensitive information from government agencies dealing with the military and the private sector; fraud through illegal access to computer data banks, including electronic funds transfer (EFT); and intentional or unintentional destruction of computer data banks.

Until recently, the need for protecting information with the science of cryptography has primarily been of interest only to the military and diplomatic communities. Only rarely had private industry or government considered the use of cryptographic techniques to protect routine business transactions. Since a significant portion of day-to-day transactions occur over the telephone system, the replacement of telephone wires with microwave radio transmissions has created a condition in which information can be intercepted without requiring a physical tap on the telephone line; for example, interception can be accomplished passively and undetected. It has been widely reported in the press that some foreign governments routinely monitor U.S. communications in major metropolitan areas with sophisticated computer systems. Information about high technology, negotiating positions, and the economic well-being of a particular industry or government agency is more useful to a foreign government than military information, particularly during peacetime.

WIRETAPPING

Most telecommunications terminate in the office, computer room, or communications center. In specialized applications, some telecommunications terminate in radio operations facilities, are mobile as hand-held equipment, or are installed in moving vehicles. Except for radio operations (but not exclusively), telecommunications terminal equipment is linked to a network by cables on the users' premises.

In a typical operating environment, cables are hidden in subflooring or in raceways above lowered ceilings. Cables serving individual rooms or offices come together in wire closets in large buildings, perhaps on each story of the building or on alternate stories, as necessary. In smaller buildings these local cables from individual terminals may come together at a point in the basement known as the cable head. Cable heads are the points, in the larger buildings as well, from which feeder cables run to the wire closets above.

Within the wire closet is a terminal block used to connect the many pairs of wires in the cable to the individual instruments in offices on that floor. Each pair of wires represents one discrete telephone number that may be assigned to a telephone, computer terminal, or other piece of equipment. The pairs are frequently labeled with these discrete numbers inside the wire closet.

Interception of data or information flowing over cables within buildings requires that the crime of wiretapping be committed. Little sophistication is required to place a tap on such cables; the materials necessary can be purchased from electronics retailers for under $25. A tape recorder connected to a carrier-operated relay switch can be added for less than $500; thus, an unattended intercept position that will operate automatically only when the targeted line is active would have been created. In most office buildings, security responsibility for wire closets and cable heads is unsettled. It is commonplace to find wire closets unlocked and standing open. Moreover, few security personnel are technically trained to search for and identify wiretaps.

It is also frequently true that telephone company installers and maintenance technicians will write the telephone numbers or extension numbers of their various terminal positions in wire closets. They do this in order to quickly find a desired wire at some future time. Such lâbeling is also convenient for a wiretapper.

From the cable head in the basement, a large *feeder cable* may run underground or on a series of telephone poles to the telephone company's central office and once there to a mainframe, which is a large automatic switch to which all subscribers served by the central office are connected. High-volume customers may have private branch exchanges (PBXs) on their premises that will switch internal calls without connection to the telephone company's central office. Organizations not equipped with PBXs may have their internal calls routed outside their office buildings to a central office several miles distant even though the number dialed is only 100 feet away.

Feeder cables, whether underground or on telephone poles, are vulnerable to wiretapping. Although a specifically targeted pair of wires serving only one piece of equipment may be more difficult to locate in feeder cable than in a wire closet, the task is not excessively complicated, and no complicated or expensive equipment is needed. The wiretapper is also spared the risk of detection while breaking and entering private property. With only a few cents' worth of copper wire the interceptor can even strap together the victim's cable and his own so that off-premise interception is possible.

Inside the telephone company's central office physical security is a relatively strong deterrent to penetration by unauthorized outsiders and even unauthorized company personnel. It is possible, however, that some telephone company personnel, with access to mainframes and other switches, could be subverted to arrange for wiretaps at the request of unauthorized persons. Recent history has shown that abuse of power has led to authorized wiretaps that were later judged to be not specifically related to law enforcement or illegal.

When the calling party dials a long-distance or toll call, he will be switched through the mainframe at the central office to the telephone company's toll switch. The call is then routed to a trunk cable serving the geographical area desired. Depending upon the telephone company involved and the two areas to be connected, the call may then be completed entirely via cable. In this case, the cable would be underground, overhead on poles, or under water. None of these cable applications is immune from wiretapping. Overhead cables provide the easiest access for an eavesdropper. No tools or equipment more specialized than those needed to tap indoor cables are needed.

Underground cables must be dug up and penetrated but, once replaced, may give the advantage of being less susceptible to discovery. Underwater cables may be categorized similarly. In the case of transocean cables, one common carrier advertises regularly in coastal newspapers that charts showing cable locations are available to commercial fishermen to help them avoid the frequent raising of cables.

MICROWAVE

The probability is greater than 70%, however, that a dialed toll call will not be transmitted entirely by cable. Cable is expensive to procure, lay, and maintain. The common carriers are increasingly using terrestrial microwave radio and communications satellites to provide toll service. These modern technologies provide the carriers with increased capacity at lower operating costs. They are, however, uniquely vulnerable to passive electronic interception. *Passive* means that the transmission can be intercepted without any physical connection to cables or transmitting equipment or without entry to private property, as in a physical wiretap. It is impossible to detect this type of interception.

Terrestrial microwave transmitters and repeaters focus a highly directional beam of radio energy toward a receiving antenna within line of sight some distance away. The distance is determined by terrain factors and ranges, typically, from 25 to 40 miles. At the transmitting antenna not all the transmitted energy is radiated along the path of the focused beam. Surrounding the transmitter are side and back lobes of the same signal aimed 25 to 40 miles away. These lobes are irregular in shape and size but normally extend in a roughly circular pattern several miles around the transmitter. Interception of any of the lobes or the main transmitted beam, either between the transmitter and receiver or behind the receiver, will permit an eavesdropper to gain access to the information or data contained on one or more of the channels.

Interception and exploitation of a terrestrial microwave signal are more complicated than for wiretapping. Some knowledge of radio engineering is needed; and up to $5,000 worth of antennas, receivers, and demultiplexers is required. Telephone companies usually load a microwave signal with as many simultaneous conversations and data streams as possible. An eavesdropper would have to sort through as many as 14,000 simultaneous channels to find the target. This is a more troublesome feat to accomplish, since toll calls dialed in a public-switched telephone network will be transmitted on unpredictable channels. For example, two identical long-distance calls, placed 5 minutes apart, from the same caller to the same party will probably be transmitted on different channels, selected by the carrier's computer on the basis of traffic load.

If the calling party's organization has leased private lines or a privately switched network from the carrier, calls will always be switched over those private channels and are much more easily located by the eavesdropper. The Federal Telecommunications System (FTS) is a privately switched network operated for the federal government by a common carrier. All FTS calls are switched over private channels and are more easily located by an eavesdropper. The channel assignments made by the carrier for the FTS network are static and represent some simplification of the targeted call search for the interceptor.

COMMUNICATIONS SATELLITES

Toll calls spanning the longest distances may be routed over communications satellite circuits. These calls are switched to a satellite earth station and may travel over cables or microwave circuits between the caller's location and the transmitter. The satellite link consists of an up link beamed from the transmitting earth station to the satellite in synchronous orbit above the equator. The satellite receives the signal and transmits a signal called the down link back to earth. Up links are similar to terrestrial microwave transmissions. They are in the same general frequency range, are highly directional, and are characterized by the same side and back lobe phenomena. Interception of the up link requires either hovering in an aircraft somewhere in the beam path or positioning a receiving antenna on the ground, within the lobes, somewhere near the transmitter. The former is impractical, and the latter carries a high risk of detection. An interceptor is unlikely to attempt either because of the particular vulnerability of down links.

A communications satellite transmits a signal specifically intended to be received at many points by many different receivers over a broad area, perhaps thousands of square miles. The down link antenna pattern is referred to as a footprint. A single footprint can cover the entire United States.

Unauthorized listeners, appropriately equipped, can point a satellite receiving antenna at the satellite and receive the down link anywhere within the footprint. A call placed from California to Maine by satellite may be interceptible in Texas or Michigan. Detection of the interceptor is extremely unlikely and almost impossible. An investment of less than $10,000 in equipment would be necessary for the interception, and its operation would require the expertise of a highly trained technician. Interception resources of this magnitude are beyond the reach of all but the most serious and qualified adversaries.

RADIO FREQUENCY

The fourth transmission technology (in addition to cable, microwave, and satellite) is radio transmission. Voice or data signals introduced into telecommunications systems may be switched into radio systems at frequency ranges far below those used for microwave signals or satellites. It is commonplace for calls to originate or to be placed to parties in motor vehicles, boats, or aircraft. Telephone companies maintain very-high-frequency (VHF) radio facilities for this purpose. Also available from other operators is a patching service in which telephone calls may be handled over long distances via high-frequency (HF) radio.

The vulnerabilities of VHS and HF radio systems to interception differ from those for microwave signals. HF transmissions may be carried over thousands of miles, and VHF transmissions may be intercepted up to 50 miles away. Radio transmissions of this type are omnidirectional, unlike those of microwave transmissions and satellites, which tend to be focused beams of radio energy.

INTERCEPTION RISKS

Thus, each of the four technologies used by telephone companies for the transmission of calls is vulnerable to interception. For the purposes of this discussion, telephone companies are any telecommunications common carriers providing voice, record, or data service. Companies that lease teletypewriter terminals and equipment may not provide any voice service, but their transmission methods will be identical to those of voice telephone companies. The same is true of newer companies organized to furnish computer network connections and companies offering facsimile, electronic mail service, or teleconferencing. Regardless of the type of traffic originated by the customer, the transmission technology used must be one of the four previously described. A summary of interception risks is provided in Table I.

Larger carriers may have all four technologies available within their corporate resources. Smaller companies may have fewer, as in the case of those serving small geographical areas with cable only. *Value added* carriers own no transmission facilities but lease and resell them from larger companies. The vulnerability of major carriers' systems is shared by value added networks (VANs). When a carrier has more than one technology available, it can normally choose which one will be used to connect different points at different times. Under FCC regulations, however, a subscriber may specify a desired mode of transmission, which must be provided by the carrier, if available. An additional charge is made for the exercise of this option.

The growing concern expressed by users of telecommunications about the privacy of their traffic has prompted a few of the common carriers to offer protection. In only one case, however, has a carrier publicly offered to assist subscribers in arranging end-to-end protection from one office to another. A number of satellite carriers have plans to encrypt some of their channels. In these cases the entire link used by a subscriber will not be protected end-to-end; the tails be-

TABLE I—Interception risks

Transmission Technology	Interception Method	Detection Probability	Approx. Cost	Level of Expertise*
Radio (HF/ VHF/UHF)	Passive electronic interception	Unlikely	$100	1
Cable (wire)	Wiretap on premises	Fair	$25	2
Cable (wire)	Wiretap on telephone company premises	Good	$500	2 to 3
Cable (wire)	Wiretap on cables not on user's or telephone company's premises	Poor to Unlikely	$500	3
Cable (fiber optics)	1981—None known	Unknown	Unknown	Unknown
Terrestrial microwave	Passive electronic interception	Unlikely	$3,000	3
Communications satellite	Passive electronic interception	Unlikely	$8,000	3 to 4

*To estimate the relative level of expertise (training or knowledge) required for interception, the following skill levels are suggested:

Level 1:	No training or skill necessary.
Level 2:	Ninth-grade reading level and understanding of basic wiring skills.
Level 3:	Electronic technician skills with knowledge of telephone systems.
Level 4:	Electrical engineer with computer science background.
Level 5:	Computer scientist with electrical engineering and communications background.

Table copyright © 1981 by Marketing Consultants International, Inc.

tween their offices and the earth stations, which are on cable or microwave, will remain unprotected and vulnerable.

Telecommunications managers responsible for the protection of sensitive or private information and data must consider several factors when procuring telecommunications service for their organizations. Unless the manager plans to arrange for cryptographic protection himself, he should determine whether the carrier he is considering offers protection. In the absence of such an offer, which means that his organization's traffic will be unprotected, the vulnerabilities of the transmission technologies should be reviewed. If satellite or terrestrial microwave transmission is considered too vulnerable, cable routing can be specified. In terms of vulnerability to interception, cable routing is considered least vulnerable and is the preference for sensitive information and data.

GOVERNMENT POLICY

The federal government has recognized the vulnerability of nonclassified but useful information that may be routinely transmitted by electronic means. Computer data banks are of particular interest. The heads of all executive departments and establishments were directed by OMB in 1978 to develop security programs for federal automated information systems.[1] Certain responsibilities were assigned to the Secretary of Commerce, the Administrator of General Services, and the Commissioner of Personnel Management. General responsibilities for information systems' security programs aimed at eliminating or reducing threats of disclosure of sensitive data and unauthorized manipulation of official databases were assigned to the heads of the using agencies.

The federal executive branch agencies are in various stages of implementing the program outlined by the OMB circular. This program includes the following minimum steps:

1. Designation of System Security Officers (SSOs) for each federal automated information system
2. Screening of personnel whose duties require access to automated information system components to determine their trustworthiness, in accordance with policies developed by OPM
3. Incorporation of administrative, physical, and technical safeguards for each sensitive application
4. Establishment of programs for periodic audits and recertification of security safeguards for each sensitive application
5. Inclusion of security specifications for the acquisition or operation of computer systems or services, whether procured by the using agency or by the GSA
6. Assignment of responsibility for conducting periodic risk assessments for each computer installation operated by or on behalf of the using agency

Agencies that have determined that their automated information systems are being used to process or store unclassified sensitive data must institute security measures to limit access to sensitive data to authorized personnel only. Most agencies are essentially unaware of the vulnerability of data transmissions to interception by unauthorized parties. The few that are aware are characterized by an attitude of needing first to deal with the more keenly appreciated threats to physical security and sensitive-file access and manipulation. Exceptions are noted among the departments and agencies dealing with the financial world or with law enforcement, where some procurement of data encryption has taken place. Progress is slow in this area because many potential users of telecommunications protection equipment are unconvinced that threats are credible or that the protection is cost effective.

PROTECTION TECHNOLOGIES

Communications security for voice and data messages requires the use of a variety of technologies, depending on specific application requirements. In voice communications there are two primary techniques: (1) scrambling of the analog voice signal and (2) converting the analog voice signal to digital form and then implementing any of a variety of digital encryption techniques by using standard cryptographic technology. Voice scrambling and digital voice encryption techniques have distinct advantages and disadvantages.

Voice scramblers offer a significant human-factors advantage in that they can provide excellent speech quality and speaker recognition. However, this is offset by the fact that the level of security for voice scramblers is considered limited when compared with the strength of digital encryption techniques. Digital voice encryption systems offer a significantly higher level of protection at the expense of speech quality and speaker recognition. Digital voice systems use devices that create artificial speech at the receiving end, thereby creating an unnatural-sounding voice.

Usually, voice scrambler systems offer short-term protection, wherein the length of time that a message can be considered protected is measured in minutes and hours. However, with digital encrypted messages, length of protection (security) is measured in terms of person-years of effort to decipher.

In data communications applications (computer-to-terminal, terminal-to-terminal, and terminal-to-computer), security principles are identical to those employed in digital voice systems, except that data systems are not burdened with the operational concerns of voice systems, these being intelligibility and speaker recognition. The vast majority of messages likely to be routinely intercepted are digital communications, since the eavesdropper can automate interception and message analysis. Nevertheless, high levels of security for digital communications may be achieved relatively easily.

There are no standards for establishing levels of security, at least for now. Consideration must be given to the sophistication of the security device and key management, along with the sophistication and resources available to the potential eavesdropper. This requires a case-by-case evaluation.

Basic levels of protection are usually defined as *tactical or strategic*:

Tactical protection is used to restrict the information from an observer or listener for a period of time measured in minutes or days. A variety of simple techniques are available that provide this level of protection at a reasonable cost.

Strategic security involves a situation in which the eavesdropper would require long periods of time to decode the message before receipt of useful information. Strategic security is used to protect information from interceptors who have sufficient resources (adequate funds and state-of-the-art computers) to decipher messages in periods measured in months and years.

PROTECTION PRODUCTS

In considering the optimum solution to communications protection problems, careful consideration must be given to the type of commercial products available. As of February 1982 a total of 39 vendors offering 185 different products were identified. In order to be listed as a vendor the requirements were as follows: (1) off-the-shelf availability, (2) a published specification data sheet, (3) published price lists, and (4) a stand-alone product.

A *stand-alone product* is defined as a device that is operationally independent of other devices and merely plugs into an existing communications system. In the case of voice communications, the device is plug-compatible with existing telephones, RF systems, or data devices; the unit interfaces with a typical data communications modem. In a few cases the vendor offers a total integrated system—for example, FAX encryption, wherein the protection device represents only a small portion of the complete system. These systems are also included if protection is a standard feature.

In order to simplify the product/technology categories, the listed products are categorized by application areas:[2]

VS-A —Voice scrambler analog devices
VE-N —Digital voice encryption devices operating at 2,400 bps or less (narrowband)
VE-W—Digital voice encryption devices operating beyond 2,400 bps (wideband)
DE —Data encryption devices

Of the 39 vendors, about one-third represent nondomestic suppliers. Even though there are twice as many domestic vendors as foreign, the total number of products from each technology segment is about the same. The product mix by domestic and nondomestic vendors is illustrated by Tables II and III.

TABLE II—Communications security domestic vendors (February 1982)

Vendor (Total: 82)	Product	VS-A	VE-N	VE-W	DE
American Satellite Corp.	1				1
Analytics	5				5
Boeing Aerospace Company	1	1			
Codex Corporation	1				1
Collins Communications	3				3
Collins Telecommunications	7	5	1		1
Com/Tech Systems	4				4
Controlonics	13	13			
Datotek	14	6		1	7
Extel	1				1
Fairchild Electronics Co.	1				1
FARGO	2	2			
GTE Sylvania	2		1	1	
Harris GCSD	1		1		
Harris RF	6			6	
IBM	2				2
Lear Siegler	1	1			
Linkabit	1				1
Mieco	12	12			
Motorola, Inc.	7	1		6	
Ocean Technology	1			1	
Racal-Milgo	2				2
Rapicom	3				3
Scientific Radio	1	1			
Technical Comm. Corp.	11	8			3
Transcript International	1	1			
Totals	104	51	3	15	35

Source: "Who, What, & Where in Communications Security"[2]
Table copyright © 1981 by Marketing Consultants International, Inc.

TABLE III—Communications security nondomestic vendors

Vendor (total: 13)	Products	VS-A	VE-N	VE-W	DE
AB Transvertex	5		1	1	3
AEG Telefunken	5		1		4
BBC Brown Boveri	9	6	1		2
Crypto AG	13	2	1	2	8
Gretag	13	2	2	1	8
Marconi C&B	1	1			
MSDS	9	2	2	1	4
Merck & Hollander	1				1
Miller Comm. Ltd.	1	1			
Racal-Datacom	12	6		1	5
Tadiran	3		1	1	1
Telsy	6	6			
Teltron	3	3			
Totals:	81	29	9	7	36

Source: "Who, What, & Where in Communications Security"[2]

REFERENCES

1. OMB Circular A-71 (Transmittal Memo No. 1). "Security of Federal Automated Information Systems." July 27, 1978.
2. Nye, J. Michael. "Who, What, and Where in Communications Security." Hagerstown, Maryland: Marketing Consultants International, Inc., April 1981, updated February 1982.

IBM information network performance and availability measurement

by RICHARD C. SOUCY and RICHARD M. BAILEY
IBM Information Network
Tampa, Florida

ABSTRACT

A key requirement in the network service business is that specified service levels be managed to the end user's satisfaction. The capability to measure and report end user response time and availability is essential. This paper will describe measurement techniques that were developed to track these important service level attributes in the IBM Information Network (IBM/IN). These techniques apply to most complex SNA networks.

IBM INFORMATION NETWORK PERFORMANCE AND AVAILABILITY MEASUREMENTS

The IBM Information Network (IBM/IN) was primarily designed to provide local attachment of various processor and terminal types for access to remotely located terminals and processors. This was mainly accomplished with the use of IBM Products, for example 3705s, NTO, 3032s, and ACF/VTAM Rel. 3. The resulting SNA network consists of centralized processors and multitiered and remote 3705 multiplexors, as illustrated in Figure 1. Since this paper is concerned specifically with performance and availability, design philosophy discussion will relate to these service levels only.

REQUIREMENTS

The performance and availability requirements given to the technical community were defined as viewed by the end user. Regardless of what entry city the end user connects to, the end user must experience the following service levels:

1. User response time (90th percentile) shall not exceed a specified number of seconds for each of three defined transaction classes per application, namely trivial, intermediate, and complex.
2. System availability shall be no less than specified percentage between 90 and 100% during business hours, as defined by external announcements. It shall include the reliability effects of components between the end user and the target application.

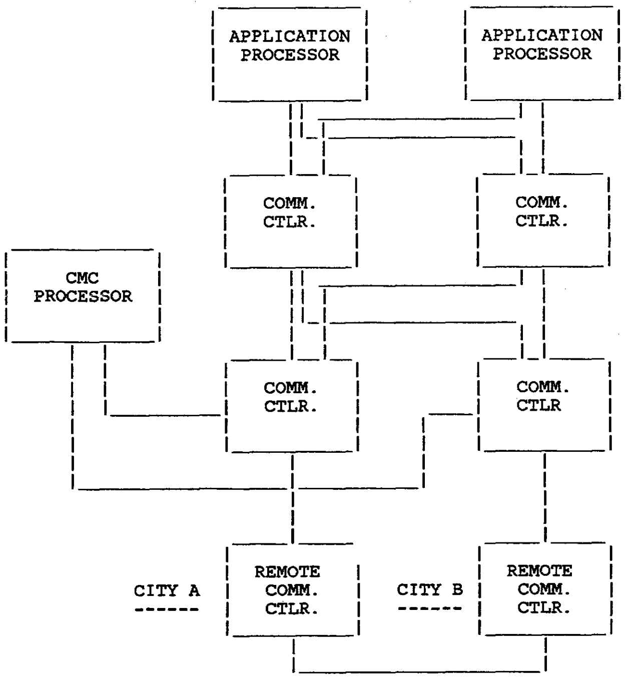


Figure 1—Sample of the route configuration from application processors and the CMC processor to remote cities

PERFORMANCE OBJECTIVES

Given the requirements, the system was divided into three parts:

1. The processors containing the target applications to which end users wish to logon.
2. The network, which ranges from the channel connecting the processors to the remotely located 3705.
3. The local line connecting the user control unit to the remote 3705.

Using SNAP/SHOT simulation and analytic techniques with forecast load values, a network specification was derived and written to specify a transit delay objective that if not exceeded would insure the end user requirements. It was felt at that time that the network delay was the critical specification, not only because it was the most complex and required detailed analysis but because it also had to be correct for the sake of the lead times required to upgrade links if needed. Similar techniques were used to evaluate the processors and local-line capacities to assure that their aggregate, including network transit delay, did not exceed the requirements.

AVAILABILITY OBJECTIVES

Using information about the reliability of the processors, operating systems, 3705/NCP, links, lines, and so on, a system availability specification was written to establish an end user availability satisfying the requirement. The end user availability was derived as the product of the processor availability, including the application and system programs, the network availability, and the local line connection. Of particular interest was the derivation for the network whose availability depends not only on the individual components but also on the various primary and alternate paths that SNA provides between the end user and the desired application. The application is considered unavailable if all paths to a destination are unavailable. Another key network component is the communication management configuration (CMC). In an SNA sense, the CMC *owns* all resources in the network. It is a processor dedicated to the management of the network and receives and stores the error status of these resources. The CMC also receives all end user logon requests to any application, and it controls activation and deactivation of all resources. Because of its importance to end user availability, redundant backup devices and procedures were implemented to meet the end user objective.

AVAILABILITY REPORTING

The measurement and reporting of availability in a complex system can be a rather difficult and expensive undertaking. There is general lack of tools for this purpose, so a significant effort to obtain comprehensive and accurate measurements is required. What follows is only one facet of the measurement and reporting process within the IBM/IN. A variety of reports, ranging from the availability of individual components to the impact of outages on the user population have been developed. Given the appropriate outage data, the production of availability reports is straightforward. However, in a system configured of multiple processors that can be accessed by remote locations through several paths, the problem of outage determination is not so straightforward. This is a task that does not lend itself readily to manual procedures. Manual procedures are not usually accurate, therefore it was always the intention to automate fully the functions of gathering, reducing, and reporting on availability data.

The first problem, then, is where the outage data will come from and how will it be collected. There are a number of system logs and a variety of system messages that provide some information on the status of system components. However, the standard information is incomplete and the data are found in different logs on all the processors. Gathering the data from available sources is a particularly messy logistical job, considering that additional processors may be added in the future and require more logs. Therefore, an availability data collection system had to be developed that not only collects all pertinent data at a central point but also generates data not otherwise available.

The CMC was chosen as the central control and logging point, and all traffic in the network used for reporting is forwarded to the CMC. The function of the CMC may be moved to another processor owing to its failure or planned outage, so the availability data collection system had to be designed to run on the backup processor and use the log local to the backup processor as the central collection point. This system, which not only collects the data but also executes the commands that generate the data, is called the Network Programmed Operator (NPO); this runs as a task under the network communications control facility (NCCF).

The primary control function of NPO resides in the CMC, with subsidiary components in each of the other processors. These subtasks are basically the same, so that as processors are added the subtask is incorporated in NCCF in the new processor with no additional development. The primary NPO component establishes sessions with the subsidiary components; availability data arriving at the processors is forwarded to the CMC over these sessions to be saved in the central log. Some of the functions executed at each processor are

1. The examination of unsolicited error messages and command response messages
2. The time-stamping and transmission of selected messages to NPO at the CMC
3. The processing of commands sent by the CMC for execution at that processor
4. The establishment and maintenance of timers for the execution of timer-driven commands

The collection of network status information requires that the NPO subtasks in the application processors be always in session with the main task in the CMC processor. These sessions are automatically established by the main task when the network is activated. These sessions, like any other sessions, are subject to loss due to failing network components or processors. To insure against the loss of status data, the main NPO task recognizes when a session has been lost and reestablishes it automatically once the failing component is again operative.

The availability of the network components of the system is the most difficult to determine accurately. The difficulty lies in our network design, which allows up to five alternate paths from each remote node to each application processor and to the CMC processor. If the primary route fails, communications are reestablished on one of the alternative routes. Therefore, a remote node and path to the remote node are considered available if the communications controller and any one of the defined paths are operative.

As can be seen in Figure 1, the remote cities *A* and *B* can access both the CMC and the application processor through several paths. The network availability could be determined from standard message data produced by the operating system. This would require that all status and error data be mapped against a database defining the resources and the connectivity of the resources of the network. Apart from the complexity of reducing the data in this manner, there is also the problem of keeping the database current as the network configuration changes.

A more desirable method, in our view, is to obtain a snapshot of the status of all the routes to a remote node when a status change occurs on any one of the routes to the node. This status is obtained through the use of a command that will test the operability of all routes from a processor to a designated location. Since this path test command adds network traffic, it is desirable to execute it only when needed, that is, when a status change occurs. The determination of when to issue the commands is based on the use of a subset of the standard network messages selected as triggers. Trigger messages are responses to commands to activate or deactivate resources, messages indicating the successful recovery of a resource and error messages reporting a resource failure. The CMC owns all network resources, so all trigger messages for the network are returned to the CMC. Other networks can be attached with resources that are not owned by the CMC; however, the NPO subtask resides in the processor that owns the resources and it is in session with the main task. The subtask forwards messages of interest from the attached network to the CMC.

The NPO tasks intercept each message that is returned to the processors to determine if it is a type that is in the trigger list. When one of these messages has been sensed, the CMC will issue a route test command that tests all the routes to the remote location indicated by the trigger message. The test command requires the designation of the route to be tested, by virtual route, explicit route number, or class of service name. When class of service name is used, all routes listed in the class of service table under that name are tested. This allows this mechanism to be independent of changes in the route destinations of the network. At the same time that the CMC issues the route test, NPO instructs subtasks in the other processors to execute the route test to the same remote node.

Responses to route tests originating at the CMC arrive there directly, while those originating at other processors are forwarded to the CMC and recorded in the log. The test provides the status of all paths from each processor to a remote location. This data, collected for a 24-hour period, is reduced to give an accurate picture of the outages on all network paths for the period. The status of network paths is maintained across collection periods so that data from a previous period coupled with new data shows a continuous picture of network availability.

Formal reports are prepared each day detailing network availability for the previous day. Another report shows the average network availability for the previous 30 days and a 30-day rolling-average report provides availability trends.

These reports, as illustrated in Figures 2 and 3, provide a picture of how the network is being managed to service-level objectives. What is also needed is the availability of access to an application for the user. This involves not only the network availability data but also the availability of tail circuits to the user location and the availability of the application and the processor on which it resides. Tail circuit outages are derived from unsolicited error messages and responses to activations and deactivations pertaining to a line. Application and processor outages are determined from messages indicating loss of a session with a processor and periodic sampling of the status of the applications. These data are also accumulated in the central log and provide the basis for an exception report showing the user's availability to any application where that availability is less than 100%. This report, as illustrated in Figure 4, shows the availability of each part of the path, line, network, and application, and the combination of all of them. The combination of the availability percentages is computed according to the unoverlapped outages of the components.

The same data that are used for the previously mentioned reports are also used to develop additional exception reports on the availability of individual resources such as lines and communication controllers.

PERFORMANCE MEASUREMENT

Response time is another service level that must be tracked continuously. Response time directly affects the productivity of a user, and it is one of the main criteria by which a communication system is judged. Continuous monitoring of response time is necessary to alert operations and management when problems are occurring in the system. Response time is viewed in at least three ways: in a system perspective, in a

```
Report: 422-11         NETWORK AVAILABILITY REPORTING           10/27/82
            NETWORK AVAILABILITY TO BOUNDARY NODE (DAILY)
-----------------------------------------------------------------------
DATE: 10/26/82                 JULIAN:  299           24 HOURS
ORIGIN: CMC PROCESSOR TAMPA  SUBAREA: 001
-----------------------------------------------------------------------
DESTINATION                                        NUMBER      AVERAGE
               SUB        NCP          PERCENT       OF      LENGTH OF
    CITY       AREA       NAME        AVAILABLE    OUTAGES   OUTAGE(hrs)
------------- ---- -------------- ----------- ---------- -----------
atlanta        045     ibmn6rt        100.00        0          .00
chicago-1      043     ibmttrt         98.68        1          .46
dallas         041     ibmdsrt        100.00        0          .00
houston        037     ibme7rt        100.00        0          .00
los angeles    039     ibmwgrt        100.00        0          .00
new york-1     048     ibmf3rt         99.43        1          .40
philadelphia   047     ibmmbrt        100.00        0          .00
san francisco  040     ibmn0rt         99.99        2          .00
tampa-1        049     ibmwhrt        100.00        0          .00
tampa-2        038     ibmflrt        100.00        0          .00
wash d.c.-1    046     ibmlcrt        100.00        0          .00
```

Figure 2—Network availability report (sample data only)

```
REPORT: 422-35        NETWORK AVAILABILITY TO BOUNDARY NODE
                 30 DAYS AVERAGE FOR THE PERIOD ENDING 10/27/82
-----------------------------------------------------------------------
ORIGIN: CMC TAMPA  SUBAREA: 001      24 HOURS               10/27/82
=======================================================================
DESTINATION                                       TOTAL      AVERAGE
               PERCENT    OF      LENGTH OF   LENGTH OF  TIME BET.
    CITY      AVAILABLE OUTAGES  OUTAGE(hrs)  OUTAGES    FAILURES
------------- --------- -------- -----------  ---------  ----------
atlanta           99.59     4        .56         2.24      129.79
chicago-1         99.81     4        .30         1.20      130.20
dallas            99.63     3        .62         1.86      153.13
houston           98.91     5       1.11         5.55       89.67
los angeles       99.12     3        .68         2.04      141.44
new york-1        99.96     1        .25          .25      336.00
philadelphia      99.32     2        .71         1.42      224.97
san francisco     99.84     1        .44          .44      336.00
tampa-1           99.76     1        .52          .52      334.45
tampa-2           99.93     1        .46          .46      336.00
wash d.c.-1       99.87     1        .41          .41      336.00
                  -----    ---      ----      -------   ---------
 avg. all cities  99.59    23        .55         1.19      231.69
```

Figure 3—Network availability report (sample data only)

```
REPORT: 450-21          END TO END AVAILABILITY              9/21/82
                         PRODUCTION SYSTEMS   (daily)
-----------------------------------------------------------------------
DATE: 9/20/82    JULIAN 263          24 HOURS
-----------------------------------------------------------------------
CITY
    PROD.                  LINE    NETWK.   APPL.   END TO  NBR.  CUSTOMER
    HOST   APPL.   LINE    AVAIL.  AVAIL.   AVAIL   END     OF
    NAME   NAME    NAME     %       %        %       %      OUT.
    ------ ------ ------ ------- ------- ------ ------- ------ --------
atlanta
  cmc     netm2   n6861   99.46  100.00   99.83  99.29     2    #3641
  mvs-a   appl-a  n6861   99.46  100.00  100.00  99.46     1      "
  mvs-a   appl-b  n6861   99.46  100.00  100.00  99.46     1      "
  vm-a    appl-c  n6861   99.46  100.00   98.86  98.32     2      "

chicago-1
  cmc     netm2   tt0al   99.91  100.00   99.83  99.84     2    #2921
  cmc     netm2   tt121   98.06  100.00   99.83  97.89     2    #2866
  mvs-a   appl-a  tt0al   99.91  100.00  100.00  99.91     1    #2921
  mvs-a   appl-a  tt121   98.06  100.00  100.00  98.06     1    #2866
  mvs-a   appl-b  tt0al   99.91  100.00  100.00  99.91     1    #2921
  mvs-a   appl-b  tt051   98.06  100.00  100.00  96.46     1    #2866
  vm-a    appl-c  tt051   99.91  100.00  100.00  99.91     1    #2921
  vm-a    appl-c  tt051   98.06  100.00  100.00  98.06     1    #2866

dallas
  cmc     netm2   dslal   98.88  100.00   99.83  98.71     2    #6123
  mvs-a   appl-a  dslal   98.88  100.00  100.00  98.88     1      "
  mvs-b   appl-e  dslal   98.88  100.00  100.00  98.88     1      "
  vm-b    appl-d  dslal   98.88  100.00   99.26  98.14     2      "
```

Figure 4—Exception report

user's perspective, and in the user's perception. The user's perception of response time is something that can be determined only from the level of complaints received by the service desk. Measurement from a system point of view—that is, how the individual components of a system are performing—provides the information needed to tune the components.

The most important measurement, which will show how the components of the system are working together, is delivered response time. In a so-called *star network* configuration the performance of the network component is more easily understood. This depends for the most part on the speed and type of facilities and the number of active devices supported by them. In a *mesh network* configuration, traffic is flowing in various directions through the nodes that can cause bottlenecks not present in point-to-point facilities. This is particularly true when an alternative route is used that is also the primary of another node.

Delivered response time (to the end user) is extremely difficult to measure without intelligence in the terminal device, not only to record transaction times but also to record the type of transactions from which it resulted. There are many devices and systems on the market that attach to user lines or terminals and record transaction times. However, the results of these measurements do not provide sufficient information to determine whether objectives are being met. Without a specification of the transactions and the response that should be expected, the results do not present a clear answer to whether response times are good or bad.

The objectives, as mentioned before, define three classes of transactions—trivial, intermediate, and complex—for each application. These transactions vary somewhat in the amount of network traffic they use, but the largest difference is in the amount of processor time and DASD accesses. The objectives also specify the maximum time in which the 90th percentile of a class of transactions for a particular application must be completed.

Response time measurement within IN is then directed toward determining if these objectives are being achieved throughout the operational period. The approach that was taken and is currently under development is to employ a device that acts as if it were a user executing a standard set of transactions.

A scenario of transactions within each class, trivial, intermediate, and complex, for each application was defined. These transactions are executed periodically from various remote locations and the results are sent to a central location for storage and reporting. To obtain these results, some intelligence is required at each of the remote locations. Since the basic approach to measurements has been to eliminate manual intervention, it is necessary that all the required functions be performed automatically. The advent of the inexpensive personal computer (PC) with its ability to communicate with other systems has made it cost justifiable to locate these devices at remote sites for measurement purposes. Since the objective is to eliminate manual intervention, the PC must operate unattended.

Each PC at a remote location operates independently, as if it were another user of the services competing for system resources. In order to avoid the addition of line resources at the remote node, the PC accesses the system through dial

```
      TSO END USER TRIVIAL RESPONSE TIME   LOCATION = DALLAS   12/25/82
R    |==================================================================
E    |
S    |
P    |
O    |
N    |                                 *
S    |                              * *
E    |-------------------------------*----------------------------------
     |                            **    *****
T    |              **        ****           **
I    |           ***  *                        ****
M    |      **  *      *     *                     ***
E    |    **  **           **                                ***       *
     | ***              ***                           *******   *******
S    |*
E    |__________________________________________________________________|
C    |8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24|
                               TIME OF DAY
        NUMBER OF SAMPLES = 64
        90TH PERCENTILE = 2.6 SECONDS     AVERAGE =  2.2 SECONDS
```

Figure 5—On-line report showing daily response time at a given location

```
        ******* 90TH PERCENTILE RESPONSE EXCEEDS SPECIFICATION *******

      CMS END USER TRIVIAL RESPONSE TIME   LOCATION = ATLANTA  12/25/82
R   | ==================================================================
E   |
S   |
P   |                             ****
O   |                            *    *
N   |                           *      ***
S   |                       ****           *
E   |---------------------------------------**--------------------------
    |                                         ********
T   |                                                 ***
I   |           ******
M   |      **  *      *    *                             *
E   |    **  **        ****                               *****        *
    | ***                                                      *******
S   |*
E   |__________________________________________________________________|
C   |8   9   10  11  12  13  14  15  16  17  18  19  20  21  22  23  24|
                            TIME OF DAY
       NUMBER OF SAMPLES = 64
       90TH PERCENTILE = 4.2 SECONDS     AVERAGE =  3.6 SECONDS
```

Figure 6—On-line report showing a location where response time has exceeded specifications (sample data)

facilities. This is done by directly connecting the PC to an auto-dial modem. Using dial facilities also allows the simulation of different types of terminal attachment, such as start, stop, and full screen, through the SDLC and BSC facilities.

The basic cycle of operation starts with a timer interrupt at the PC. The network is then accessed through the auto-dial modem. The PC accesses each application in turn and executes the transaction scenario for each application, storing the response times at the PC. When all transactions have been executed, the system accesses an application at the host processor that is designed to accept the data for the session. The PC transfers the response time data to the host processor and waits for the next timer interrupt to start again. Accumulated data can be viewed on line at any point, and exception reports are produced for areas failing specifications.

Since the ability to operate unattended is key to the success of this method of measurement, a comprehensive set of recovery procedures must be developed. The PC program must be able to handle the variety of problems that may be encountered on the lines and the host applications. Maintenance is also a major consideration in the design of the PC program. Any data at the PC that are subject to change are maintained in tables; when changes are required, the host program passes the new tables to the PC.

Figure 5 and 6 present two samples of the on-line response time reports produced by the system.

PERFORMANCE MANAGEMENT

A reporting/measurement system provides the data necessary to manage the service levels of a large and complex system. The reporting function measures how well objectives are being met, and it can provide insight for possible improvement. Management of service levels however, begins with requirements, design cost trade-offs and operational procedures.

Reports are formatted for both the system programmer and senior management. Although it is automated, the measurement system requires the intelligent cooperation of all the participants.

Designing and managing an SNA network for growth

by S. M. SCHIFFMAN
IBM
Tampa, Florida

ABSTRACT

The IBM Information Network is a nationwide System Network Architecture (SNA) network that offers network services and application offerings to end users. This paper describes the techniques used to design and manage the growth of the network. Emphasis is placed on the design process and the migration and implementation plans to expand the network while still providing 24-hour operation 7 days a week. Areas covered include adding processors, bandwidth, multiplexers, and end user devices.

PREFACE

The task of designing and managing a complex telecommunication network involves extensive advanced planning. The disciplines of performance and availability analysis, line cost evaluation, network topology, and distributed processing must be employed. The IBM Information Network came into being in 1982 and offers a variety of network services and application offerings to its customers. The network is designed to accommodate a variety of device and system attachments and to be able to grow as the need arises. The following discussions explain the objectives of the network and how the design has been able to meet them. In addition, the flexibility of IBM system network architecture (SNA) is highlighted by using examples of network expansion strategies.

NETWORK OBJECTIVES

There are many details required to define a network, but a clear set of guiding principles are necessary to make design decisions. In this section, we discuss the major objectives of the IBM Information Network.

Provide Nationwide (U.S.) Coverage

It is obvious that covering the entire U.S. in a short time is not feasible. Therefore, a design is required that permits constant orderly growth into new cities and geographical areas. This objective contains the main requirement for planning and managing growth. Each extension to the network has to be designed without hindering normal operation.

Allow Any Location Access to Application Processing Center(s)

Initially, the processing center exists in one location, so traffic must be efficiently routed to it. The main access into the network and subsequently to application processors is from remote concentration points. End users at these locations must have access to multiple systems. Currently, there are two types of IBM 370 operating systems to access: the multiple virtual system (MVS) and virtual machine (VM) systems. Dynamic access to these systems is provided by use of the multisystem networking functions (MSNFs) of SNA. These features will be discussed later.

Allow Any Location Access to Any Other Location

In addition to providing access to an application-processing center, it is necessary to allow an end user at any remote location to access any other location in the network. For example, this would enable a terminal user in Los Angeles to communicate with an application processor in New York. This network capability is utilized by the Network Services offering of the IBM Information Network. Highlights of this offering will be used as examples for network growth features.

These objectives were the main elements to be satisfied; the subsections that follow list a set of secondary areas to be addressed.

Availability

In order to provide quality service to end users, various techniques have to be devised to minimize outages due to network growth, moving equipment to new locations, and failures in equipment or environment. This objective has become another main theme guiding the expansion of the network.

Attachability

The IBM Information Network has to provide a wide range of device attachment. The network supports IBM 3270 displays, SNA terminals, SNA host systems, start/stop terminals and binary synchronous remote job entry work stations and host systems. These devices and systems are afforded local access to the IBM Information Network by attaching them to remote concentrators. This permits connection by cost-effective local dial calls or short-distance leased lines.

Flexibility

Designing a network before the location and the use scenarios of end users are determined provides a challenge. Different terminals, systems, and bandwidth requirements are expected to grow in an unpredictable fashion. Therefore the network has to be designed to handle different mixes of terminals and increasing bandwidth while remaining in full operation.

Response Time/Throughput

When all of these objectives are under consideration, adequate performance has to remain at the forefront. Therefore, transit delay has to be kept at a minimum even though high bandwidth may not be required for early traffic.

With these objectives in mind, the network design, management strategies, and implementations that follow will become clear. But, first let us look at the overall network design,

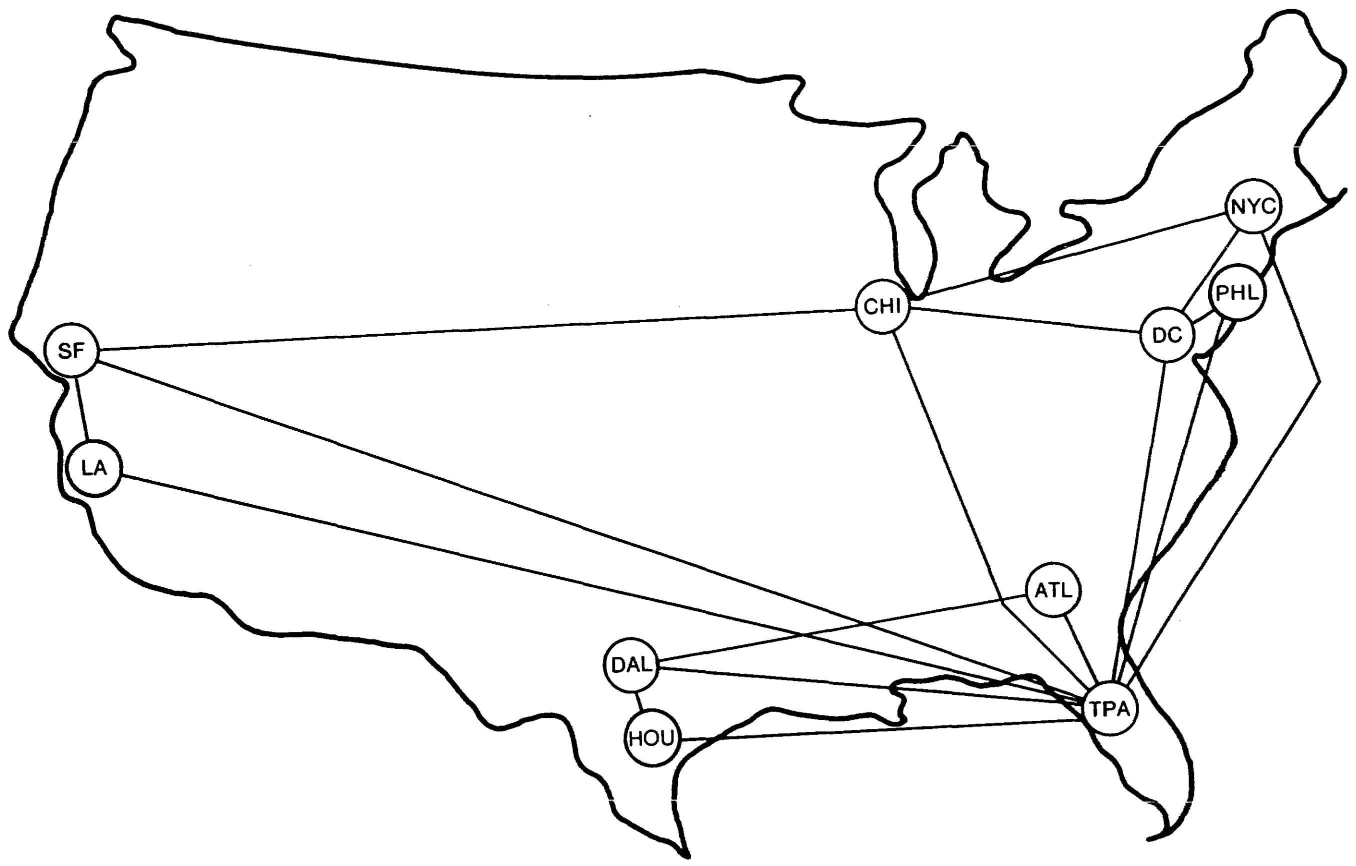


Figure 1—The IBM Information Network: geographical distribution

dissect each portion of the network, and show how designing for growth is accomplished.

OVERALL NETWORK DESIGN

The design of the IBM Information Network is based on the communication management configuration (CMC) concept, which uses the latest developments in SNA. The main features employ a central-site system to control the network for operations, problem determination, and status. The system used is MVS with the Virtual Telecommunication Access Method (VTAM) in a System 370 and the network control program (NCP) in a series of 3705 communication controllers all combined to operate a multisystem network. In Figure 1 the network topology shows a set of ten remote concentration points linked together with high-speed lines. These concentration points provide local access to end users. The topology shown was built up over a period of time and will be extended in the future.

The CMC concept is based on the premise that concentrating control of a geographically distributed network is more efficient than distributing control. As such, the network definition and migration/growth plans are centralized and kept track of on an inventory database. This provides better control over ordering and tracking network components such as telecommunication lines, modems, terminals, communication controllers, and telephone rotaries. It also provides a better perspective on the whole network, thus enabling some economies of scale to be exercised. The topology is modeled to allow for grouping of traffic onto fewer and larger high-speed lines. Modems can be ordered in larger numbers to take advantage of quantity discounts.

Not only migration and growth, but also the day-to-day network operational complexity must be controlled. The latest tools available with SNA—the Network Performance Analyzer, Network Problem Determination Aid, Network Communication Control Facility, Terminal Access Facility, Routing Table Generator, and so on—are used to plan and operate the network. As the network expands, a point will be reached where control has to be subdivided to deal with the sheer number of components. The CMC concept allows this to be accomplished by logically dividing network control initially and later planning for additional CMC processors if necessary.

The network is controlled from one location in order to concentrate network management expertise. This also provides close control by monitoring use to determine capacity planning. In addition, programs are used to automate many tasks, such as network initialization, backing up lines that

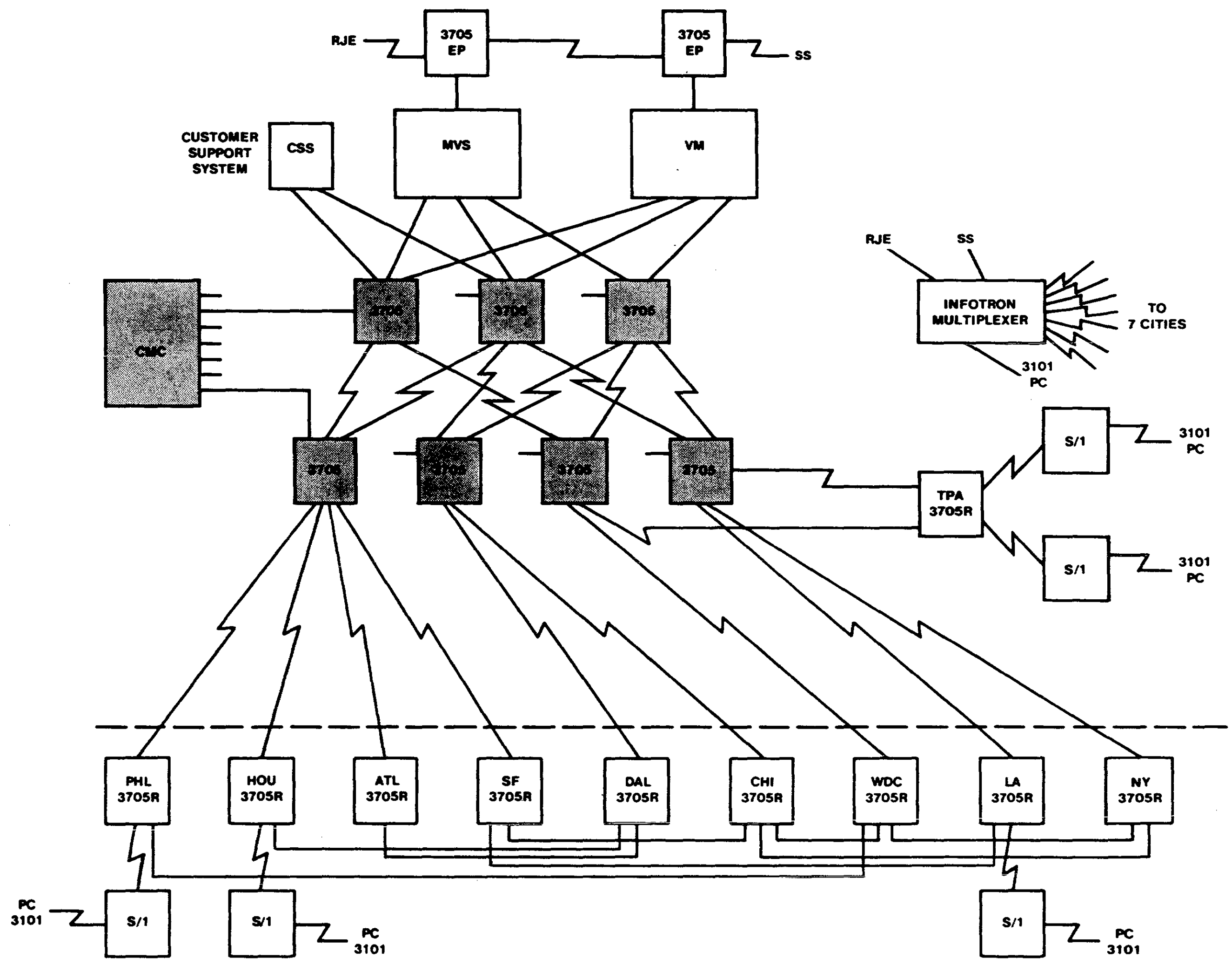


Figure 2—The IBM Information Network: network configuration

have failed, collecting statistics on trends, and responding to requests that do not require operator intervention. The strategy is to automate more and more operator tasks as they are identified, so that human error will be minimized as the network grows.

Figure 2 depicts the network configuration, including the CMC and application processors.

An application-processing center may reside at several locations as the need arises. Each one is composed of one or more processors capable of running operating systems such as MVS or VM and is connected to a switching network by a set of 3705/NCP multiplexers. The IBM Information Network initially contains a single processing center. Expansion may be accomplished using SNA MSNF features.

THE SWITCHING NETWORK

The switching network area of Figure 2 is shaded for conspicuousness. It consists of a processor that provides communication management through VTAM and network management tools. This CMC processor is linked to the network by a series of 3705s in a meshed configuration. These multiplexers provide the paths for remotely attached end users to access each of the application processors. The 3705s are configured in two forms; one set is optimized for the maximum number of channel adapters, and the other uses many high-speed (56-Kbytes/sec) lines. These meshed multiplexers form an insulator between the application processors and the remote network. All end user devices are attached to remote multiplexers, where the most frequent network changes occur. The switching 3705s are more stable and are influenced by adding application processors and/or new remote 3705s.

Expansion of application processors is accomplished by using a channel adapter connection on a local 3705 and defining the new processor to selected network components (see Figure 3). Once all channel adapters are used, additional 3705s are added by meshing one into the switching network. Advanced planning is necessary to minimize the impact of adding new processors. Keeping in mind the 24-hour operation re-

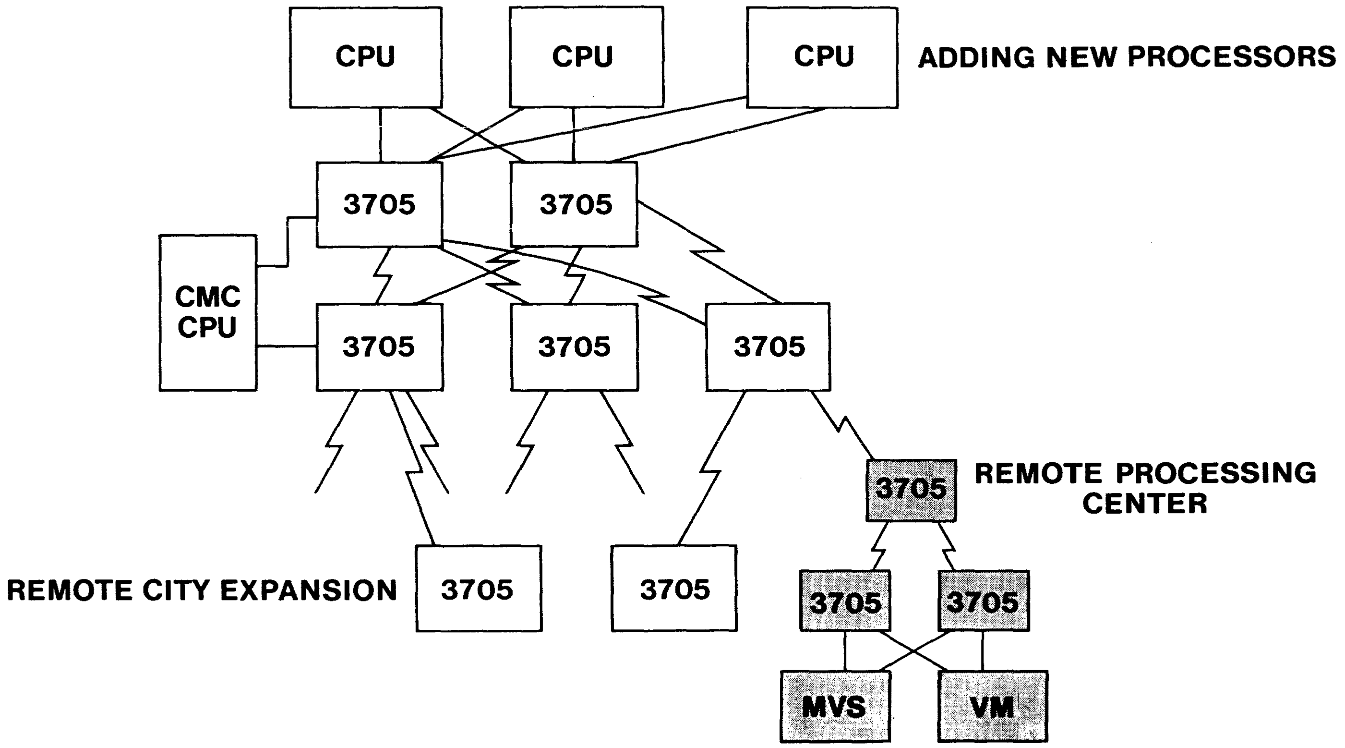


Figure 3—Switching network expansion for new cities, new CPUs, and remote processing centers

quirement, the new processor definitions are gradually added to NCPs and VTAMs. This is possible because the absence of the processor is ignored by the network even though it is defined. In addition to locally attached processors, MSNF offers the capability of attaching application processors to the switching network at remote sites. Their attachment is similar to adding remote multiplexers. In the shaded area of Figure 3 a remote processing center is depicted with application processors and communication controllers attached to the switching network. The overall network design is preserved by continuing to attach terminals to remote 3705s. Once again the rudiments of a switching network are taking form at the remote location, with a series of 3705s used as switching controllers to the application processors.

As more remote multiplexers are added in new cities, they are connected to the switching network via high-speed digital lines. Initially this is a simple process; ports already available on the second tier of 3705s in the switching network are used. The affected adjacent 3705/NCPs may be gradually updated before the new 3705 and line are added. After enough new remote cities are added, the second-tier high-speed ports will finally be exhausted. This requires the addition of more 3705s. A 3705 is first logically added to the configuration, and all the necessary paths and physical connections are determined. Once again, adjacent NCPs are gradually updated and then loaded. When all affected components are synchronized the new 3705 is physically added to the switching network. In this manner the switching network may be expanded while still maintaining normal operation.

It should be noted that all remote locations are defined with a primary route and at least one disjoint alternate route to access the application processors (see Figure 4). This definition permits 3705s to be redefined and reinitialized with updated NCPs while the network still operates normally. While reinitialization is in progress the alternate route is used automatically. Once reinitialization is complete, the primary route is available. The SNA alternate-route capability is used as a migration tool for adding new 3705s to the second tier of the switching network and increased bandwidth between the switching network and remote sites. As the network grows, other uses will be found for this migration tool.

BANDWIDTH EXPANSION

Lines between the switching network and the remote multiplexers are prone to the widest variation of use. As the end user population grows, the load increases on these lines. The transmission group (TG) line feature of SNA is exploited to add necessary bandwidth: A new line is defined and then installed between 3705s; This line is added to a TG and testing is insured by deactivating other lines for short periods. Once the testing is successful, the new line is a full-fledged member of the TG. TGs are also used for adding interim analog lines when digital service is not yet available in an area. In this case the analog lines are included in a TG and are discontinued once the digital lines are added and tested. This process usually requires 30 days to insure that the new line has stabilized.

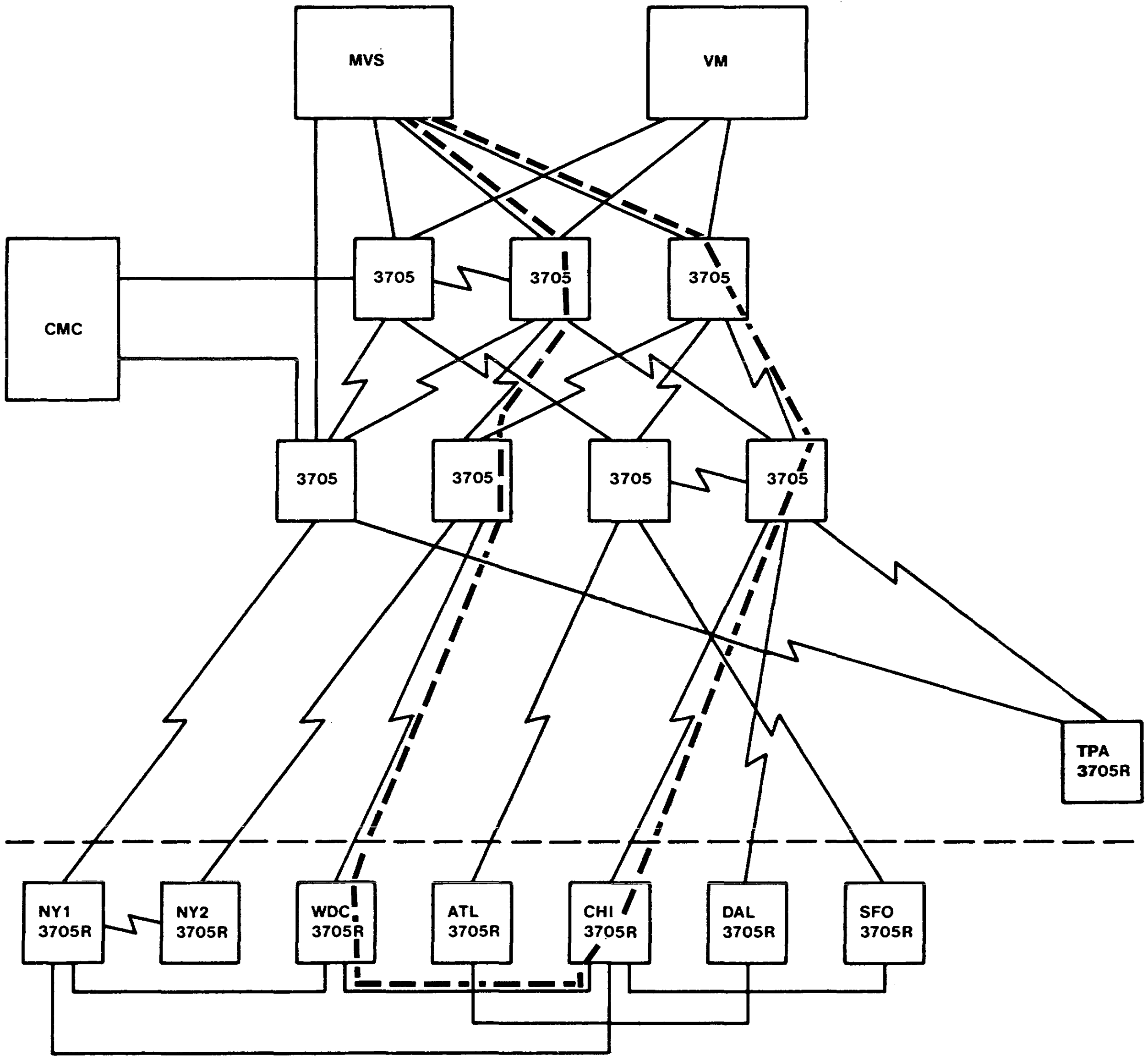


Figure 4—Alternate routes

Since experience has shown that line availability varies by area, TGs are defined with multiple lines even where no immediate plan to add lines exists. This aspect of planning ahead is a constant theme for network growth. It enables lines to be added with a minimum of disruption. In this way the use of TGs as a migration tool has become predominant.

REMOTE CONCENTRATION

As previously mentioned, all end user devices are attached to the IBM Information Network via remotely deployed multiplexers. At each remote concentration point several types of multiplexers are employed. One or more 3705s connect SNA and BSC 3270 devices and systems. Series/1s are used to perform protocol conversion of IBM 3101 and the IBM Personal Computer to full-screen emulated operation. When port capacity is exhausted, additional remote 3705 multiplexers may be added to the network and the Series/1s in turn attached to the 3705s. Both switched and leased connection are possible. An end user choosing switched connection may dial into any location supported by the network. Each remote location is managed from the central CMC site using SNA tools. Both the 3705 and the Series/1 are initialized by the CMC network operator. The remote 3705s are loaded with the configuration from a database maintained on the CMC system. The load module is transmitted to the 3705 over the telecommunication lines in the network. The Series/1s are loaded from diskettes resident on each system. This loading is triggered by remote control from the CMC site.

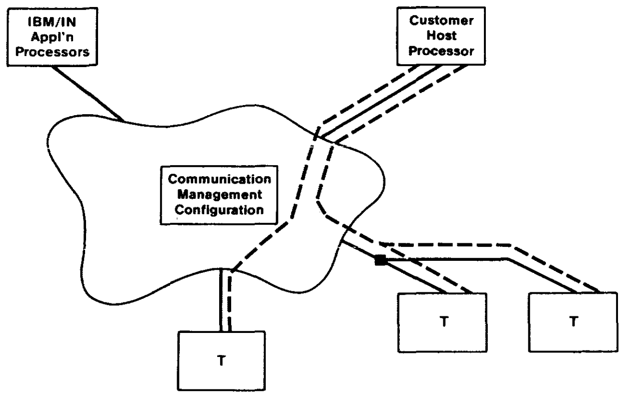


Figure 5—Network Services

A strategy of updating these configurations no more often than once a month has been chosen. The justification for this approach is based on the availability of a leased line usually being greater than 30 days. Therefore, once a leased-line end user has to be added, waiting 30 days is acceptable. The switched-line user may attach to the network almost immediately, since adequate ports with modems and telephone terminations are planned in advance. This also permits future leased-line users to attach via switched lines in the interim. Managing the switched ports requires constant monitoring of use so that rotaries, modems, and ports may be allocated for future expansion. Using network tools, connection information on dial port usage is retained. These data are used to project 30 to 90 days ahead, so that the ports may be expanded. The lead time for this activity is critical, since close coordination between the common carrier, the modem supplier, and the IBM Information Network support groups is required.

In addition to connecting end user terminals that access IBM Information Network application processors, the remote 3705s also provide connection for Network Services users. In Figure 5 a typical connection configuration is shown. Several terminals in different cities are connected to a customer-owned application processor. This connection traverses the network through remote 3705s on an SNA session. A session is initiated by communicating the request to the CMC processor. After the necessary security checks and user identification are completed, the end user is connected to the appropriate processor. Use information about sessions is collected at the remote 3705 and relayed back to the CMC for later, batch processing. In this manner, performance information can be consolidated for the remote 3705s even though both cross-network traffic and IBM Information Network application processor traffic are included. This information is reduced and used for determining when a communication line to a customer host must be upgraded to meet line load and response time objectives. In addition, data on the use of the high-speed switching network lines are also collected. Decisions on adding cross-network lines and upgrading the speed of existing lines are based on these data.

Projections are also made for the short and long term. Since digital circuits often have a long lead time before installation, sometimes more than a year, this advance planning is important. When a line has to be upgraded or added, the strategy used is the same as for adding connections between the switching network and remote nodes. The 3705/NCPs are gradually updated and lines are added to existing TGs to minimize network disruption.

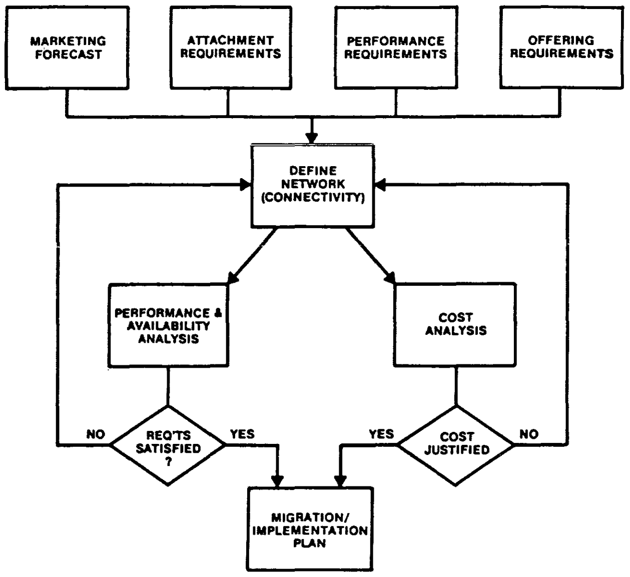


Figure 6—The planning cycle

SUMMARY

The design and management of an SNA network providing for growth require advance planning and a well-trained technical staff versed in modeling, performance analysis, topological design, measurement, and cost evaluation. These skills must be integrated in a design team that considers both short- and long-term network growth problems. These range from methods of expanding dial rotaries to major network expansions and restructuring. The process is continuous and iterative. The flow chart in Figure 6 depicts this process as applied to the IBM Information Network.

The areas that require special design for growth include

1. Expanding service to new geographical areas
2. Addition of ports for end user equipment
3. Expansion of bandwidth
4. Inclusion of more CPU power for applications

All of these requirements must be met while maintaining 24-hour uninterrupted service to end users and reasonable network transit delays.

The features of SNA permit many other flexible expansion and operation strategies. In the future, additional CMC configurations may be used to distribute network operation and to improve availability further. This extension may be accomplished by remote deployment of processors attached to the switching network. Thus, by distributing both network management and backup capabilities, long-range goals may be met.

Backup and recovery in the IBM Information Network

by K. BHADRA and S. M. SCHIFFMAN
IBM
Tampa, Florida

ABSTRACT

The IBM Information Network is a nationwide system network architecture (SNA) network that provides users access to application offerings and network services. This paper describes the design of backup and recovery methods for insuring that the availability of the network is maintained. The communication management configuration (CMC) is highlighted and the uses of current SNA capabilities to provide dynamic switching are shown. In addition, operational and human factors in achieving and managing backup are described.

THE NETWORK

The IBM Information Network is a multi-tier network controlled by a communication management configuration (CMC); it is based on system network architecture (SNA) 4.2.[1] A controlling ACF/VTAM[2] based on its MSNF[3] feature resides on a dedicated processor and serves as CMC. The first tier gives connectivity of different application processors to the CMC, and the second is the CMC tier, which links all boundary nodes at different cities with the CMC. Both tiers are channel-attached to the CMC. CMC is the owner of all network resources except for the applications, which reside in different hosts. All resources are known to CMC using the MSNF feature of ACF/VTAM.

Service Manager, an application program run under the CMC host, controls the session initiation requests of all the network users. Boundary node 3705s are connected to the CMC tier by high-speed links. Each remote boundary node 3705 has at least two direct or indirect links to the CMC tier, which ensures a completely disjoint route to the CMC from remote 3705.[4]

The MVS and VM productivity systems are channel-attached to the application tier. The application processors are configured in pairs to ensure backup in case of their failure.

BACKUP STRATEGY

The network backup and recovery[5] strategy uses most of the backup and recovery features of ACF/VTAM and NCP[6] V1R3 wherever possible. Before we describe the implementation of backup and recovery it will be worthwhile to talk about the strategy that drives the design and implementation of backup.

The design of backup and recovery in IBM/IN is directed to achieve high system availability for all IBM/IN resources, from each remote node to each service. Cost effectiveness of the backup, availability gain, and performance were evaluated continuously.

To achieve the goal of high availability in such a complex network, we have concentrated on each component. Since the CMC is the most important component of the network, a lot of emphasis has been placed on the backup and recovery of the CMC host.

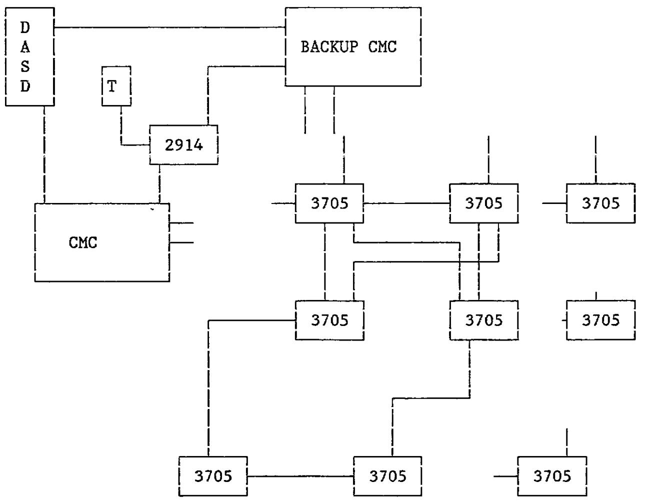
A network component failure can be either long- or short-term failure. In a long-term failure, a failing component cannot be repaired or replaced quickly. Hardware failures are good example of long-term failure, because the duration of failure is unknown. Short-term failure is a situation in which a failing component can be repaired or replaced quickly. Transmission group (TG) failure is a good example of a short-term failure that can be backed up by quickly alternate routing. In IBM/IN 10 minutes is considered to be the boundary time between a short-term and a long-term failure.

CMC BACKUP

The CMC host in IBM/IN is crucial to the availability of the network. It serves as the brain and heart of the network. To ensure that the role and function of the CMC can be duplicated within a very short period of time in another processor a set of procedures has been developed. Coordination of software and hardware between the CMC and backup CMC was an important aspect of the design.

Hardware Consideration

An MVS application processor (E system) has been designated as the backup host of the CMC (A system), and in turn the CMC is the backup of the MVS application host. The network and hot-line operator's terminals are local to CMC; through a switching system they can be locally connected to the E system. DASD required for network management are channel attached to both systems and sharable. Figure 1 shows the connectivity of the DASD and the operator's terminal in the IBM/IN.



T ... Network Operator Terminals

Figure 1—CMC and backup CMC connectivity in IBM/IN

What is Backed Up

Users of the IBM/IN are usually connected via Service Manager (SM). SM prompts users to enter desired application, then logs on to the application. When the CMC fails, all the sessions are lost, but some sessions in the logical level remain undistrubed. The backup CMC backs up all SSCP sessions that are supported by the primary CMC. Users who have lost their sessions will have LOGO on their terminal wherever appropriate, and will log on to their application when the backup CMC takeover is complete. Hot-line (customer assistance) and operator terminals are usually the first to get their session back.

Recovery and restart of a session in an ACF Release 3 environment can be either disruptive or nondisruptive. There are two basic functions in ACF Release 3 necessary for backup and recovery option.

1. Cross-domain session continuation.—In the generation of NCP all leased-line SDLC devices have been coded with ANS-CONT option (automatic network shutdown). This allows leased SDLC terminals to remain in session in a cross-domain session with an application in another host even if the owning CMC host has failed. There is no such support for SDLC dial, BSC, or S/S devices. In case of CMC failure all dial SDLC, BSC, and S/S devices will lose their session.
2. Nondisruptive takeover or restart.—There are only few SNA devices that can support nondisruptive takeover or restart if they are attached to a leased line. The following list shows some devices which support the error recovery program (ERP). The ERP-supported leased-line devices will not lose sessions when we switch back the CMC function to its original processor during switchback.

1. 3274 1C,31C,51C (configuration support B or C is required at the appropriate EC level)
2. 3271 models 11, 12
3. 3275 models 11, 12
4. 3276 with RPQ 8K1030
5. 3601,3602,36058K1030
6. 3684
7. 3776 models 3, 4
8. 3777 models 3, 4
9. 3776 (PU only)
10. 4701
11. 8100 DPCX Rel 1 modification 4 feature #6001 level C
12. 8100 DPPX FEP C
13. System/32/34/38

Backup CMC Design

The design of the backup CMC considers the following points. The backup CMC is running all the time with the same set of software (MVS, ACF/VTAM, NCCF, CCICS/SM, etc.) that runs in the CMC host, but it does not own any device outside of its original domain so long as the primary CMC is in charge of the network. The files and libraries that are required by the primary CMC to run the network reside on the channel-attached DASD and are shared by the Backup CMC. NCPs are GENed with ANS = CONT and ISTATUS = ACTIVE. As a part of VTAM initialization the application major nodes indicates ISTATUS = ACTIVE.

Initialization of the network consists of initialization of the primary CMC as well as the initialization of the backup CMC. The backup CMC remains waiting ready to back up a failure. Since the CMC host is also a backup of the application host, a set of procedures has been set up to back each other up.

CLIST and Command Automation

The backup procedure takes too many commands for it to be possible to type them in quickly in a crisis. To make the procedure user friendly, all individual commands were grouped together either in a CLIST form to be entered through NCCF terminal or in a OPERDR Job to be entered at the system console. To save time, the network operator is provided with a console of CMC, backup CMC and an NCCF terminal grouped in one location.

Procedures

The procedure is setup on a contention basis. The order of initialization of the host is very critical. In the start option of VTAM, the following are activated in the primary CMC in the initialization mode.

1. Path table
2. CDRMs
3. CDRSCs
4. Application major nodes
5. Switched major nodes
6. Local major nodes
7. Channel-attached local NCPs

These NCPs have no boundary function and have no terminals attached. In the initialization mode of the backup CMC and also in the backup mode of the primary CMC, all the items of this list except for switched major nodes with different inputs are activated as a start option of VTAM.

CMC initialization procedures start local terminals, NCPs, and CDRMs in the primary CMC. When the network is up, the E system is initialized as backup by activating local terminals, links, local NCPs, and CDRMs. One important point is that we did not activate the boundary node NCPs to set up shared ownership from the backup CMC. Our experience with the network and hot-line operator was that it detracted from user friendliness to have two sets of status of the same device in two different consoles. At the same time we were not saving more than 15 seconds by setting up joint ownership.

Most of the backup time comes from the activation of the devices of the NCPs. We have carefully ordered the sequence of activation of the remote NCPs so that loads on the links are well balanced throughout the network during activation. This load-balancing saves time and uses NCPs effectively during crisis period.

Operation

Since the CMC and backup CMC are backing each other up, one is always in the backup CMC mode while the other in the CMC mode. At the start time the CMC host (A system) is initialized as a CMC and the E system is initialized as a backup CMC in backup mode. In the event of CMC failure, the backup CMC procedure is invoked and the E system serves as a CMC as well as a productivity system. During the day when the A system is fixed, it is initialized as a backup CMC to back up the E system, which is the active CMC. If in the same day the CMC (E system) fails again, the backup CMC procedure is invoked in the A system. Otherwise, for an orderly takeover of their original roles, a switchback procedure is invoked at a scheduled hour after prime shift. During switchback, we force a failure in the backup CMC, switch resources to their original host, and initialize the E system as the backup. The probability of simultaneous failure of A and E system is very small probability, and it has not been covered in these backup procedures. For CMC hardware failure there is no problem determination time, the failure is always considered as long term, and backup is initiated immediately.

Backup of CMC on the E system, as implemented and tested, takes only 10 minutes to provide logon at every terminal of IBM/IN.

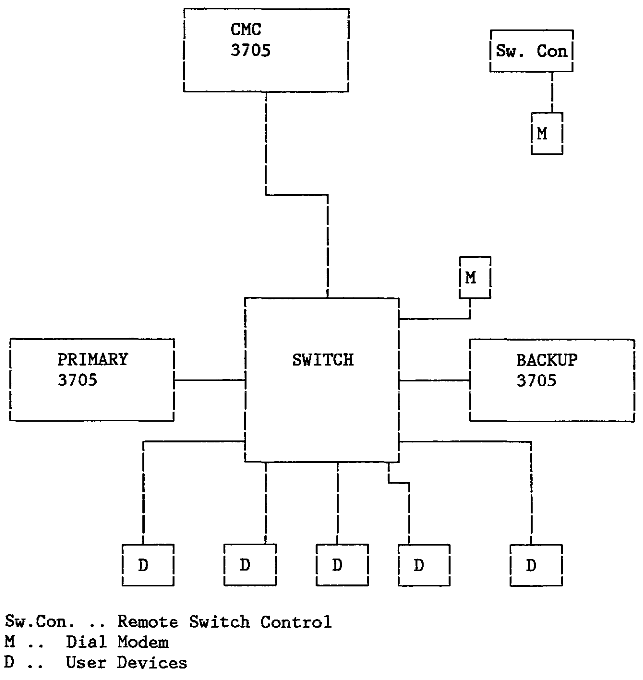


Figure 3—Backup 3705 connectivity in IBM/IN

3705 BACKUP

To increase the boundary-node 3705 availability, we have implemented two-stage backup. In the first stage we provide dial backup for leased and dial users. The two-wire dial backup for leased-line users is shown in Figure 2. In the second stage of backup, a standby backup 3705 is provided at the remote cities. The backup 3705 is connected to the production 3705 through a remotely operated matrix switching system. In the case of remote-city 3705 failure, all the lines are switched from the production 3705 to the backup 3705 and are loaded with the same GEN to activate the session. There is a 15-minute limit to this 3705 backup. The connectivity is shown in Figure 3.

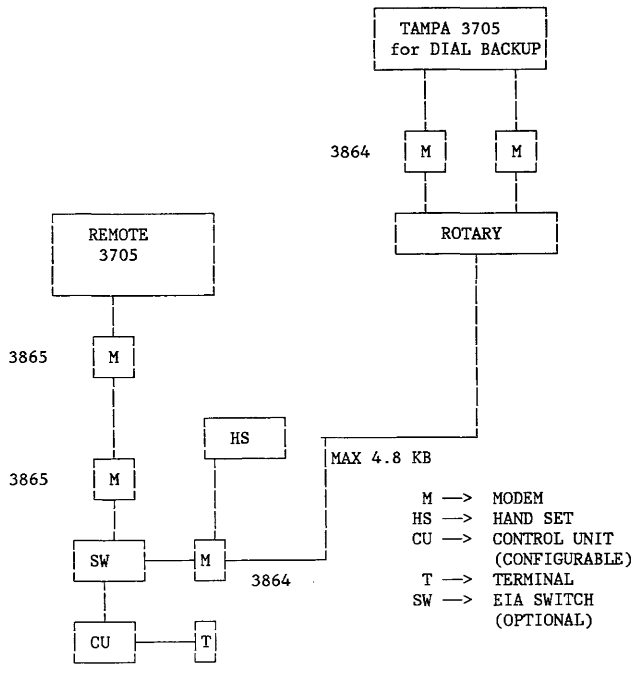


Figure 2—Two-wire dial backup using IBM modems

LINK BACKUP

Links are the arteries of a network. To back up links we exploited transmission groups (TGs) between two connecting nodes; TGs are standard features of the ACF/NCP V1R3. If any link of the TG fails, the traffic is automatically routed to the next available link in that TG. As it is not guaranteed that links of a TG will not run together side by side, TGs remain vulnerable to natural disaster. Alternate routing has been implemented to cover the TG failure. Every remote 3705 is either directly or indirectly connected to the CMC tier by at least two completely disjoint physical links. On these links at least two completely disjoint routes have been mapped from boundary node to the CMC and also from the boundary node to the application processors. The shortest and the fastest route is the primary path, and next one serves as the backup path.

POWER BACKUP

The Tampa network complex is provided with power from two different power substations. The building is equipped with uninterrupted power supply (UPS) which can supply power for 18 minutes to the network components. There is also a standby diesel-powered generator unit that backs up UPS. A power backup procedure has been set up to deal with the situation.

ACKNOWLEDGMENTS

K. Bhadra is grateful to Mr. S. Cawn and Miss L. Colombino for their helpful advice during backup CMC design, and to Mr. R. M. Campbell, without whose help implementation would not have been possible.

REFERENCES

1. System Network Architecture Format and Protocol Reference Manual, Architecture Logic, SC30-3112 (3rd ed.), 1980. Available through IBM Publication Center, 9th floor, 1801 K St., N.W., Washington, D.C. 20006.
2. ACF/VTAM Planning and Installation, SC27-0584 (1st ed.), 1980. Available through IBM Publication Center, 9th floor, 1801 K St., N.W., Washington, D.C. 20006.
3. Gray, T. P., and T. B. McNeill. "SNA Multiple System Networking." *IBM System Journal, 18* (1979), pp. 263–295.
4. Ahuja, V. "Routing and Flow Control in System Network Architecture." *IBM System Journal, 18* (1979), pp. 298–314.
5. Cawn, S. M. "Backup and Recovery in ACF/VTAM Release 3." GG22-9261, 1981. Available through IBM Publication Center, 9th floor, 1801 K St., N.W., Washington, D.C. 20006.
6. ACF/NCP. Installation Release 3, SC30-5140 (1st ed.), 1981. Available through IBM Publication Center, 9th floor, 1801 K St., N.W., Washington, D.C. 20006.

Logical problem determination in an SNA network

by ROBERT A. WEINGARTEN
IBM Corporation
Poughkeepsie, New York
and
EDWARD E. IACOBUCCI
IBM Corporation
Raleigh, North Carolina

ABSTRACT

Until recently, problem determination on a systems network architecture (SNA) network has dealt mostly with error detection on physical network components. Logical-error detection mechanisms associated with logical-network (software-related) errors were not adequately provided until the recent announcement of a new on-line interactive package called the network logical data manager (NLDM). This paper will discuss the physical and logical network environments, logical network problems, and functions provided by NLDM for logical problem determination.

INTRODUCTION

Systems network architecture (SNA)[1] can be viewed as consisting of two networks, one physical and one logical. The physical network consists of a number of hardware nodes connected by links. The function of the physical network is to control the flow of user data to, from, and between the physical network nodes. The logical network consists of the management of protocols that support the exchange of user data. Although these two networks can be viewed separately, they interact closely since the logical network operates via the physical network.

In the past, problem determination has mostly dealt with the physical network. Error-reporting and recording mechanisms evolved from stand-alone diagnostic support on individual hardware nodes to online interactive diagnostic support of remote hardware nodes. IBM has developed such program products as the Network Problem Determination Application[2] to allow an SNA user either to receive alerts from the network nodes of pending or failed conditions or to query network nodes or intelligent modems[3] for status data.

Logical problem determination has not evolved so rapidly, although several mechanisms have been provided in recent releases of SNA. Until the announcement of the NLDM,[4] the logical-problem determination mechanisms were considered cumbersome and time consuming.

NLDM was created to provide online support enabling the user, normally the network operator or a trained diagnostician, to obtain data interactively on the logical network for problem determination. NLDM constantly collects data from a logical conversation, called a session, so that information is available leading up to the occurrence of a failure.

This paper will first describe in more detail the differences between the physical and logical networks and how they interact. Next it will discuss the problems associated with logical-problem determination and how NLDM can be used to aid in identifying logical errors.

THE PHYSICAL AND LOGICAL NETWORK

The Physical Network

The SNA physical network consists of a number of hardware nodes connected by links. The hardware nodes classified according to their functions. The functions and capabilities of each of these nodes are defined by SNA. Each node has associated with it an SNA function called the Physical Unit Services, which provides management functions for it.

The understanding of portions of the physical network is the responsibility of the System Services Control Point (SSCP). An SSCP has the responsibility for the network operator interface, communication network management interface, configuration control, network startup, network recovery, and participation in the creation of sessions.

The Logical Network

The logical network consists of entities called Network Addressable Units (NAU). An NAU represents a port through which end users may access the communications facilities. Although an NAU is a logical entity, it is assigned a unique network address that depicts its physical location in the network. This unique network address provides routing information so that user data can be directed to the correct physical node within the network. There are three types of NAU defined in SNA: the SSCP itself, the physical unit, and the logical unit.

The logical unit (LU) is a set of function management services directly supporting an application program or terminal operator end user. The LU provides the services and protocols necessary for the end user to communicate with other end users. This communication, or exchange of user data, occurs by establishing a session between the two LUs on behalf of the end users. Each session has a set of agreed-to protocols that are supported in the LU services at each session end. These protocols establish such session characteristics as data syntax and meanings, data flow methods, and other session properties that will be used during the session.

Establishment of a session also implies the use of certain physical-network entities. These include nodes and links in the path between the two session end-points and the node resources at each session end, including buffers, storage, and processing capabilities.

A session is initiated by an end user, either an application program or a terminal user. The initiator of a session uses logical names, called network names, to identify the session end users. The SSCP resolves the logical-network names into the physical network addresses assigned to the session end users.

End users, including the network operator, do not use network addresses to refer to sessions. Even the physical problem determination packages, such as the Network Problem Determination Application, use network names in communicating with the network operators. Correlation of the logical-network names to physical addresses is accomplished using the services of the SSCP.

THE LOGICAL PROBLEM

Logical problems attributable to software can be classified as either detectable or undetectable. The detectable errors will result in (a) error messages that can be logged in sequential files or displayed to the network operator, (b) failure notification to the session ends, (c) storage dumps caused by a software product's abnormally terminating a session, or (d) a combination of the above. Typically, these can be well documented and resolved since the necessary information is normally available at the time of failure.

The undetectable problems make it appear to the session ends that the network has "gone to sleep"; therefore they are more difficult to resolve. These problems exhibit no external signals or notifications. They normally occur owing to either a protocol error or the loss of a message.

Protocol Errors

As already mentioned, an SNA session follows specific, agreed-to protocols. These protocols are established at session initiation. Two forms of undetectable software protocol errors can occur.

1. Mismatch or misunderstanding of the protocols required for session initiation. This can occur owing to a program error during session setup or to a mismatch due to programming support levels of the software involved in SNA sessions.
2. Incorrect setting of protocol states by either the receiving or the sending logical unit. Each protocol in SNA has a defined set of actions that it can accept or reject, depending on the current protocol state of the logical unit.

Loss of Message

Another potential problem is the loss of a message or path information unit (PIU) within the network without appropriate notification. The loss may result in an end user waiting for a message. It should be noted that NLDM addresses undetectable errors, since SNA describes the protocols that are executed as a result of detectable errors and failures in both software and hardware.

EXISTING APPROACHES TO THE PROBLEM

Although the undetectable errors are difficult to analyze, SNA does provide facilities that support logical problem determination. However, these are considered cumbersome and difficult. Methods that can be employed include the following:

1. Activating traces on a teleprocessing link, on a transmission group,[5] on an ACF/NCP/VS resource,[6] or within a host access method.
2. Taking storage dumps of all suspected software components that are contained within the physical path used by a session.
3. Placing traps within the suspected software components to cause a planned abnormal termination when the logical error reoccurs, then taking a storage dump and handling this error as a detectable failure.

NLDM—AN APPROACH TO RESOLVING UNDETECTABLE LOGICAL PROBLEMS

NLDM collects, stores, and monitors network logical problem determination data. NLDM uses the services of the network communications control facility (NCCF)[7] to obtain and display the session related data at the central or remote NCCF network operator terminal.[8]

NLDM collects two types of session-related data: session awareness data and session trace data. Session awareness is notification by the SNA access methods—ACF/TCAM[9] and ACF/VTAM[10]—to NLDM that a session has been successfully started. The session awareness data, which NLDM timestamps, consist of a session start and end indication, session partner network names and network addresses, session type, and configuration information about the session end points. The session trace data are supplied to NLDM by the SNA access methods and by the ACF/NCP/VS program. The session trace data obtained from the host access methods include the following parts of a message or PIU: the transmission header, the request/response header, and the first 11 bytes of the request unit. Session trace data are only collected for sessions involving a resource for which a trace has been started. NLDM collects these data in storage for active sessions and places it in a VSAM file at session termination for limited historical purposes.

NLDM also collects session trace data from the boundary function serving a session end point in an ACF/NCP/VS boundary node. These data consist of the last four PIU sequence numbers, which are contained in the transmission header, and the appropriate control block information about the ACF/NCP/VS resource involved in the session. This ACF/NCP/VS data is automatically sent to NLDM at session termination or can be solicited during the session, as long as the trace for the session is active.

The collected session data is displayed by an NCCF network operator using NLDM's supplied hierarchical display structure. The NCCF network operator uses the network names to display session-related data. The NCCF operator can display session-related data for both session ends at the same display station, although two NLDMs may be used to collect the session data for a cross-domain session.

To provide the network operator with a single network operational view in an SNA cross-domain environment, an NLDM uses the services of NCCF to establish communications between itself and the other NLDMs. From the session awareness data obtained at session activation, an NLDM can determine which other NLDM to access to obtain data relating to that session. This is illustrated in Figure 1. In the figure, NLDM1 in HOSTA will be provided by SSCP1 with the session awareness data for an application whose network name is APPL.

```
HOSTA                            HOSTB
*---------------------*          *------------------------*
| APPL  |   NLDM1     |          |           | NLDM2      |
|       |-------------|          |           |------------|
|       |   NCCF1     |          |           | NCCF2      |
|-------|-------------|          |------------------------|
| ACF/TCAM or         |          | ACF/TCAM or            |
| ACF/VTAM  *-----*   |          | ACF/VTAM    *------*   |
|           |SSCP1|   |          |             |SSCP2 |   |
|           *-----*   |          |             *------*   |
*---------------------*          *------------------------*
   NCP1 |                           NCP2     |
   *-------------*                  *---------------*
   |ACF/NCP/VS   |--------/_________|ACF/NCP/VS     |
   *-------------*                  *---------------*
 N1       |                            T1  |
     *---------*                       *---------*
     |NCCF OP  |                       |Terminal |
     |Console  |                       *---------*
     *---------*
Notes: Session between APPL and T1
       Session between NLDM1 and NLDM2
```

TABLE I—NDLM display hierarchy

```
                         MOST RECENT SESSION
                         *-----------------*
                         | Session Partner |
                         | Status          |
                         | Start/end times |
                         *-----------------*
Specific       Session        |     Session         Session
Session        Parameters     |     PIU Trace       ACF/NCP/VS
Configuration                 |                     Trace
  -----------------------------------------------------------------
  |              |                  |                 |
*--------*   *------------*     *-------------*  *-----------*
|SSCP-ID |   | Protocol   |     |Time,PIU type|  |Sequence No|
|PU-ID   |   |   Used     |     |Sequence No. |  |Control    |
|local   |   |            |     |RH Indicator |  | Blocks for|
*--------*   *------------*     *-------------*  | Resource  |
                                 PIU   |         *-----------*
                                 Detail 1
                                 *-----------*
                                 |Specific PIU|
                                 |TH,RH,      |
                                 |11 Bytes RU |
                                 *-----------*
```

Figure 1—Network configuration

This occurs since SSCP1 is the owner of APPL and participates in the session set-up between APPL and T1. NLDM1 cannot directly obtain session awareness or trace data relating to terminal T1, since T1 is not under the control of SSCP1. NLDM1 must locate the appropriate resource owner and request the session awareness and trace data from the NLDM associated with that resource owner.

In this example, if the operator requires session trace data from T1, then NLDM1 will request this data automatically from NLDM2. The data will then be returned to NLDM1 for display to the requesting NCCF network operator. Note that if the two resources (APPL and T1) had been under the control of the same SSCP, then only one NLDM would have been involved in the data collection and display.

NLDM provides the user with a hierarchy of displays that help the user obtain session-related data from both session ends. Table I illustrates the NLDM hierarchy used to obtain logical-problem determination data for a specific session. The example in the second section after this illustrates how the NLDM displays can be used in solving a logical or software-related problem. To the terminal end user, this problem will appear as if the "system has gone to sleep." The example uses a bracket protocol error attributable to a software programming error. Prior to describing the NLDM methodology used to identify the undetected error, a short description of the bracket protocol will be provided.

BRACKET PROTOCOL

The bracket protocol[11] defines a specific set of rules for controlling a conversation between two logical units connected in a session. The set of bracket rules to be followed for a particular session are agreed to at the session's initiation. Rules identify who can start or end a new conversation, when it can be started, and when each LU can send data within the conversation. The use of the protocol insures that data from each session end are not sent at the same time.

The bracket protocol defines several states that one must obey in order to have an orderly conversation. Bracket indicators placed in the request/response header (RH) are used to control the bracket states. The intent of this section is to describe the bracket protocol, but only so far as is necessary to describe the logical error.

To start a related conversation, the authorized session end places a begin bracket (BB) indicator in the RH. When all the user data is sent and a reply is required, a change direction (CD) indicator will be placed into the RH. The CD indicator signals the receiving session end that it is its turn to send data. This flip/flop of the conversation controlled by the CD indicator continues until the conversation ends with an end bracket (EB) indicator.

The setting of the CD indicator is extremely important. If the receiving LU is not given this indication, it cannot send data back to the requestor.

EXAMPLE OF LOGICAL PROBLEM DETERMINATION

Let us take an example that will indicate how an undetected error can be diagnosed as a bracket protocol error. Consider a new application program called APPL that uses bracket protocol between itself and various terminals. Let us assume that a specific session exists between APPL and T1, as illustrated in Figure 1. In this example, the application program will start a conversation with T1 to ask a specific question and receive an answer. Then based on the answer, the application will start one of several related conversations to obtain more information.

Let us assume that the terminal operator has no indication that an error occurred. To the terminal operator, the logical error will appear as a long wait. This wait will continue until the terminal operator becomes frustrated and requests assistance from the network operator. At that point, the NCCF network operator can use NLDM to display the session activity associated with the terminal T1, as illustrated in Figure 2. From this screen, the NCCF operator can see that an active session is still in progress between APPL and T1 and that further work is needed to isolate the problem. The NCCF operator has several options. One option would be to display the configuration data screen associated with terminal T1 and then use the physical problem determination techniques to see whether a physical node problem caused the error.

Another alternative is to continue to use NLDM to try to find the source of the problem—for example, by using it to display the session activity associated with each session end. Let us assume that the NCCF operator displays the session related data for APPL, which is shown in Figure 3. In Figure 3, the first four entries under the SEL (SELection) column are equivalent to the following SNA messages:

1. APPL initiating a conversation with a BB indicator
2. APPL asking the T1 operator a question, then allowing T1 to respond by sending a CD indicator
3. T1 answering the question, then allowing APPL to respond by sending a CD indicator
4. APPL ending this related conversation with an EB indicator

These four messages should appear normal to the NLDM operator, since they adhere to the application scenario and the bracket protocol.

Continuing with the scenario, the application will now analyze the received data and start a different conversation to gather other information. It will use a bracket sequence similar to that used on SEL lines 1 through 4.

The data on the NLDM screen indicate that the next conversation was started on SEL line 5 with a BB indicator. This message was followed by another message sent by APPL on line 6; note, however, that no CD indicator was placed on this message. This indicates to a trained NCCF network operator or NLDM diagnostician that an error has occurred, since the terminal cannot respond without receiving a CD indicator.

Since this application program was new, the NLDM operator can assume that the application program failed to request that the access method place the CD indicator in the message for this specific conversation. In this case, further diagnostic tests will not be necessary. The application program must be fixed to request that the CD indicator be placed in the appropriate message field for this conversation.

Protocol error diagnosis will require either an NCCF operator with SNA knowledge or diagnostic procedures identified by the installation. If neither are available, then the NLDM operator can request that NLDM record this data on its VSAM file for later use by a trained diagnostician.

The case of a lost PIU is easier to detect than the protocol error, since the NLDM operator need only see a mismatch

```
*---------------------------------------------------------*
| NLDM.SESS                                     Page 001   |
|            * SESSION HISTORY FOR SELECTED NAU *          |
| NAME: T1                                DOMAIN: NCCF2   |
|---------------------------------------------------------|
|     ***PRIMARY****    ***SECONDARY***                    |
| SEL NAME TYPE DOM    NAME TYPE DOM     START TIME  END TIME  |
|( 1) APPL  LU  NCCF1  T1    LU  NCCF2 08/05 15:20:45 ACTIVE   |
|( 2) TRANS LU  NCCF1  T1    LU  NCCF2 08/04 13:20:30 14:10:04 |
|  ...                                                     |
|(12)                                                      |
|          (This line reserved by NLDM for error messages) |
| ENTER SEL- AND PT(P-TRACE), ST(S-TRACE), P(SES PARMS),   |
|           PC(P-CON) OR SC(CON)                           |
*---------------------------------------------------------*
```

Figure 2—Session history for selected NAU

```
*---------------------------------------------------------*
| NLDM.SESS                                     Page 001   |
|           * SPECIFIC SESSION TRACE DATA *                |
|--------PRIMARY---------------|--------SECONDARY----|--DOM---|
|NAME=APPL  SA=1  EL=10        | NAME=T1  SA=3 EL=20 | NCCF2  |
|---------------------------------------------------------|
|SEL TIME      SEQ  DIR TYPE   **REQ/RESP HEADER0*** RU LEN |
|-------------------------------------------------------- |
|( 1) 16:35:30 0020 P-S FM HDR  BB                  009   |
|( 2) 16:35:30 0021 P-S DATA    CD                  03C   |
|( 3) 16:36:15 0018 S-P DATA    CD                  020   |
|( 4) 16:36:20 0022 P-S DATA    EB                  01C   |
|( 5) 16:36:25 0023 P-S FM HDR  BB                  009   |
|( 6) 16:36:30 0024 P-S DATA                        03D   |
|  ...                                                     |
|(12)                                                      |
|          (This line reserved by NLDM for error messages) |
| ENTER 'R' TO RETURN TO PREVIOUS DISPLAY                  |
| (command input line)                                     |
*---------------------------------------------------------*
```

Figure 3—Specific session trace data

between the last four PIU sequence numbers on each session end. What is not easy to discover is which software component in which network node "lost" the PIU. The only configuration data available with NLDM are the physical units' networks names identifying the nodes supporting the session end points.

By understanding the complete physical route assigned to the session, the network operator could use other problem determination methods to attempt to isolate the problem. The network operator must use some combination of access method operator commands, a user-supplied application program, and a "network road map on the computer center wall" to obtain this data. Only then could the network operator use current methods, such as product traces, to further investigate where a message was lost within the network.

CONCLUSION

NLDM has enhanced network logical problem determination by simplifying the user procedures used to obtain session related data for logical problem determination purposes. This simplification reduces time spent in problem determination activities. It also makes it less likely that more drastic detection methods, such as link traces and program traps, must be used. Problem determination for these logical errors has been simplified but not eliminated.

REFERENCES

1. Sundstrom, R.J., and G.D. Schultz. "SNA's First Six Years: 1974–1980." *Proceedings of the Fifth International Conference on Computer Communications,* Atlanta, Georgia, 27–30, October 1980, pp. 578–585.
2. *NPDA Version 2 General Information Manual,* GC34-2061, IBM, 1982.
3. Huon, Simon, and Robert Smith. "Network Problem Determination Aids in Microprocessor-Based Modems." *IBM Journal of Research and Development* 25 (1981), pp.3–16.
4. *NLDM General Information Manual,* GC30-3081, IBM, 1982.
5. Gray, J.R., and T.B. McNeill. "SNA Multiple-System Networking." *IBM Systems Journal,* 18 (1979), pp. 263–297.
6. *ACF/NCP/VS Version 2, Release 1, ACF/NCP/SSP for the IBM 3705: General Information,* GC30-3058, IBM, 1982.
7. *NCCF General Information Manual,* GC27-0429, IBM, 1983.
8. Weingarten, R.A. "An Integrated Approach to Centralized Communications Network Management", *IBM Systems Journal,* 18 (1979), pp. 484–506.
9. ACF/TCAM, Version 2 General Information: Introduction, GC30-3057, IBM, 1981.
10. *ACF/VTAM Version 2 General Information,* GC27-0608, IBM, 1982.
11. *Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic,* SC30-3112, IBM, 1980.

Planning high-speed digital services in the Bell System

by GARY J. HANDLER
American Telephone and Telegraph Co.
Basking Ridge, New Jersey

ABSTRACT

High-speed digital transport has been an integral part of the Bell Network since the introduction of T1 facilities in the early 1960s. Since that time a growing percentage of short- and long-haul digital facilities have been integrated into the network via various technologies such as digital radio, coax, and fiber. More recently, customer needs for data communications have required the extension of high-speed digital transport to the customer premises.

This paper discusses the growth of high-speed digital services. We describe the customer applications that led to the requirement for such transport, the technological environment, and finally some near-term and longer-term plans for high-speed transport capabilities.

CUSTOMER REQUIREMENTS

Customer needs for high-speed digital transport arise from two types of applications. The first is applications such as voice and low-speed data, where a customer's volume requirements are so large that concentration is economical, and therefore access to high-speed transport is desirable. The second type of applications are bulk-data transfer, video, etc. These applications inherently require high-speed digital transport.

For certain applications, for example bulk-data transfer between two customer locations, private line solutions are acceptable. For others, such as video conferencing among many locations, an access to a switched network is more desirable. One potential future application, which would require a sophisticated switched distribution scheme, is Pay TV or video-on-demand. Penetration of such services into the residence market could bring a swift revolution to the deployment of high-speed access and high-speed networks.

TECHNOLOGY

The first widespread deployment of high-speed digital technology was with the 1.5 Mb/s T-carrier facility, which was introduced in the early 1960s. Today almost 50 percent of metropolitan facilities are digital. Initially, T-carrier was primarily used for transporting 24 voice band channels, but later its application was broadened to carrying high-speed data.

The next important transmission media is digital radio. This capability is being introduced into the network at a great rate. Various forms are available, for example, 20 Mb/s, 78 Mb/s, and 90 Mb/s, depending on the terminal equipment used.

Other technologies in use include coax, satellite, and finally, optical fiber. Fiber of course holds the most promise at this time—it has the potential of carrying high-speed (90 Mb), low-error-rate, digital signals for long distances with large repeater spacings. As the cost of this technology comes down, fiber right to the customer will become economical. This will allow for fiber in the distribution plant, leading to applications such as video-on-demand or two-way video in the homes.

There is also a significant amount of work being done in the area of connecting and switching digital streams. To connect 1.5 Mb/s digital streams and multiplex and demultiplex the 24 voice band channels within, D channel banks have been developed. To interconnect 1.5 Mb/s streams, including the 64 Kb/s single voice band channel level, the Digital Access and Cross-connect System (DACS) was developed. The DACS can be controlled from a terminal and allows for switching any input channel to any output channel. Thus we have a slow-connect high-speed digital switch. Various new high-speed digital switch fabrics are under consideration and these will make an appearance as required by customer demand.

It is expected that by 1990, 90 percent of the metropolitan and about 30 percent of the long-haul facilities will be digital. Analog facilities will continue to dominate the long-haul portion because they are economical for carrying voice traffic.

TWO NEW SERVICES

In this section we describe two new services, one switched and one nonswitched, for which tariffs were recently filed by the Bell System.

The first new service is the terrestrial digital service (TDS). This is a two-point private line offering of 1.5 Mb/s transport. In addition to the transport, various service options such as central office multiplexing of 24 voice channels and protection switching are available. Figure 1 shows some of the technologies that may be used in providing a two-point TDS connection. The customer sees an interface provided by the network channel terminating equipment (NCTE). Behind this interface, a wide variety of terrestrial technologies, such as T1, T4, and digital radio may be employed to provide the service.

The second new service is the high-speed switched digital service (HSSDS). This is a common-user, slow-switched digital network, currently available in multiples of 3 Mb/s. Access to this network is via dedicated access lines. Calls may be placed on a reservation basis. Switching is to be accomplished via the

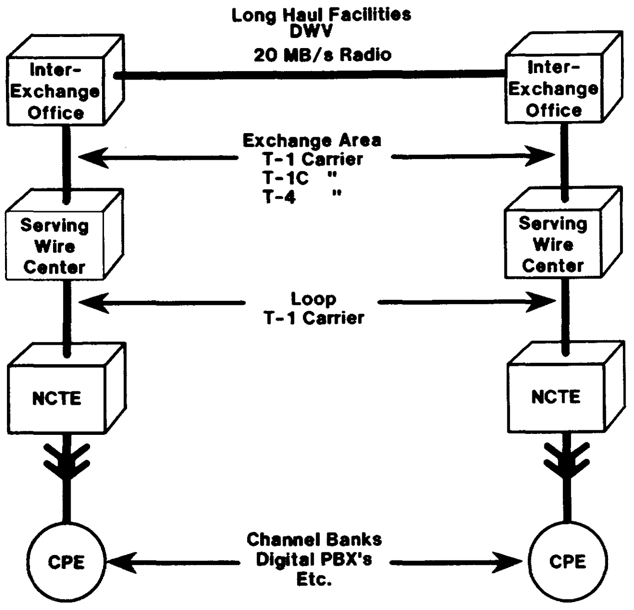


Figure 1—Typical technologies used to provide terrestrial digital service

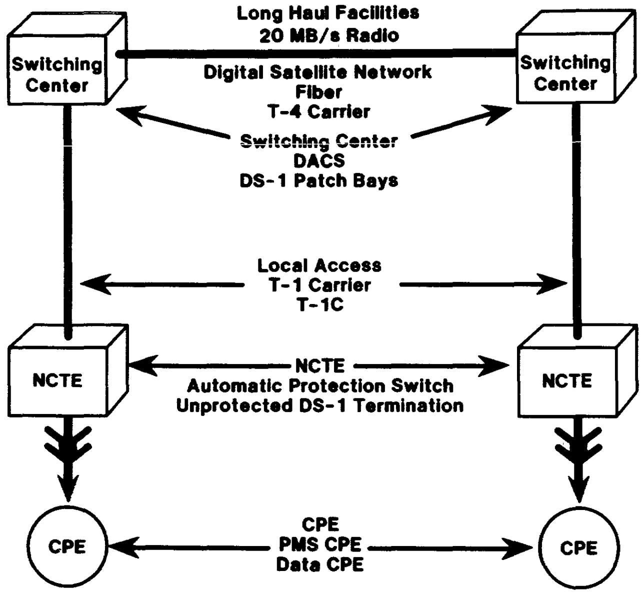


Figure 2—Typical technologies used to provide HSSDS

DACS technology previously described. The primary initial use of this service is for video conferencing. Typical technologies that may be used in the provision of this service are shown in Figure 2. Figure 3 shows the sites planned for 1984 for switching centers for HSSDS. Local access will be available from those centers. It is expected that this capability will evolve to a 1.5 Mb/s switched capability in the near future.

FUTURE CAPABILITIES

The potential for future high-speed digital services is limitless. As the cost of technology decreases and the demand and willingness to pay increases for such services as video-on-demand and two-way video, a point will be reached where radically new technology in the local loop will be justified.

Figure 4 depicts a high-speed digital communications network, useful for such applications. This particular architecture involves deploying lightguide all the way to the customer premises. Three channels may be provided, two for video in each direction and a third for other needs, such as voice and data. Remote nodes may peel off the appropriate channels to each home via wavelength division multiplexing techniques. These remote nodes achieve needed concentration prior to connecting to the lightguide feeder leading to the central office. Here new high-speed digital switch fabrics perform the necessary switching functions.

Architectures such as this are in the experimental stages in many countries. When the right combination of customer needs and transmission technologies are found, then the experimental phase will be over, and rapid deployment will follow. High-speed digital networks will be an integral part of the integrated services digital network (ISDN). That plan envisions customer access to be provided by a high-speed digital pipe, where the customer uses as much or as little of the available bit-stream as required for a particular application. Clearly the architecture described above meets the service concept presented by ISDN.

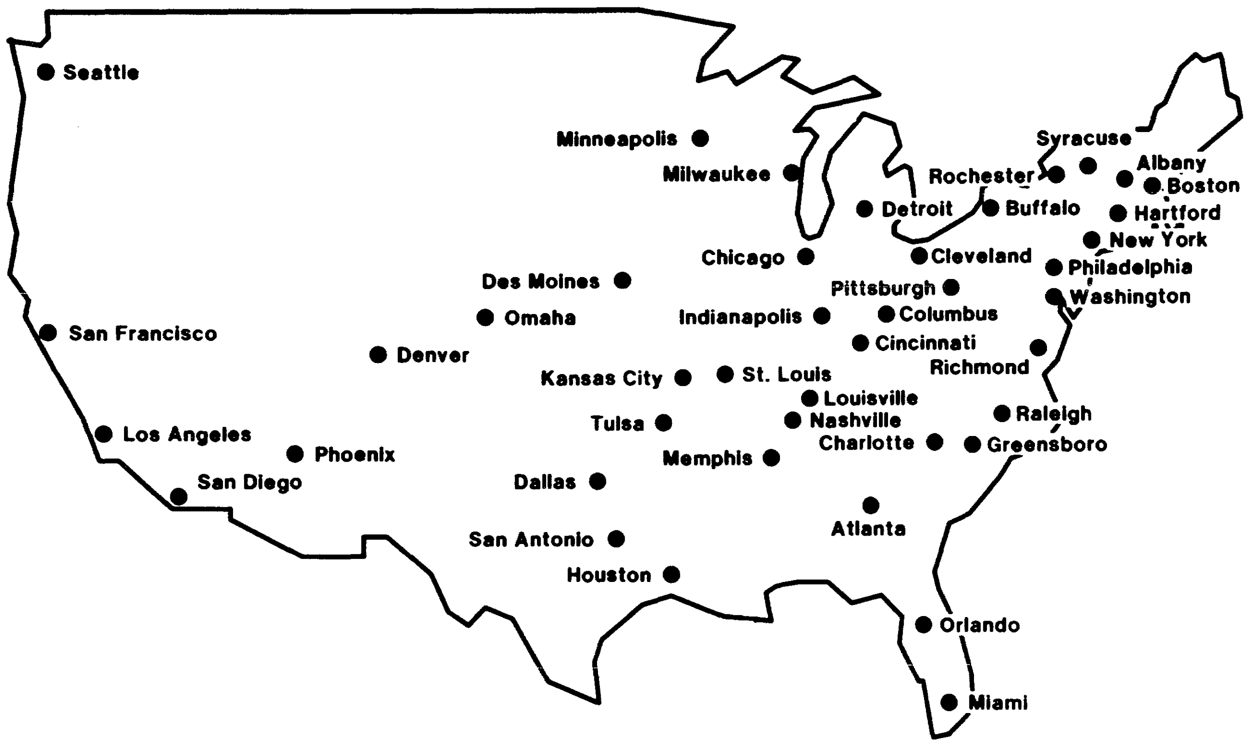


Figure 3—Planned switching centers in 1984

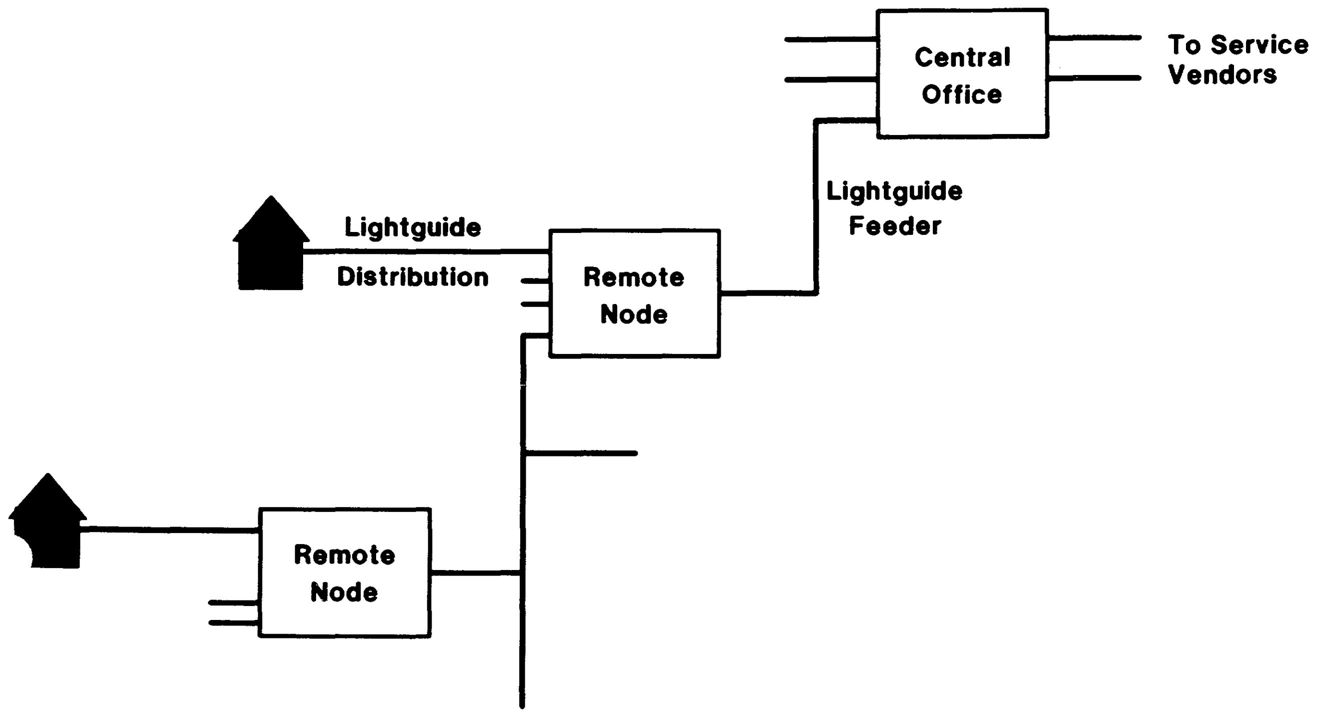


Figure 4—High-speed digital distribution configuration

Three heuristics for improving centralized routing in large long-haul computer communication networks

by IVAN M. PESIC and DANIEL W. LEWIS
University of Santa Clara
Santa Clara, California

ABSTRACT

Several algorithms have proven to be useful in computing the shortest path between two nodes in a network.[1-7] Their complexity depends on the problem definition and size of the network measured by the total number of nodes N. In these algorithms, finding all the shortest paths from a fixed node is a computation of complexity $O(N \times N)$. These algorithms have been used successfully in long-haul networks for many years. Recent growth of such networks to large numbers of nodes (e.g., $N > 500$) demands a more efficient approach. This paper develops three heuristics based on measured topological characteristics of computer communication networks and applies them in the construction of faster algorithms. Determination of the shortest path between two random nodes is then shown to require 50 to 60% less computation.

INTRODUCTION

Background

The problem of finding the shortest path in a given network is not a new one. Dijkstra simply stated the problem as:

We consider N points (nodes), some or all pairs of which are connected by a branch; the length of each branch is given. We restrict ourselves to the case where at least one path exists between two nodes. Find the path of minimum total length between two given nodes P and Q.

In the late 1950s, Kruskal[1] and Dijkstra[2] proposed the first algorithmic solutions to the routing problem. These and subsequent algorithms[3–7] all have satisfactory performance when applied to a network with a small number of nodes, but as the network grows larger, the number of necessary operations grows polynomially with N. Such algorithms are simply not fast enough for routing applications in large networks (e.g., $N > 500$). Hierarchical routing has been suggested as one solution; particular applications for networks with distributed control have appeared in the literature.[9, 10]

Dijkstra's problem definition is intentionally general and thus simplistic. If taken literally, it assumes a connected topology with otherwise random characteristics, providing only an assigned cost for each pair of nodes (with a cost of infinity signifying no connection) to guide the routing algorithm. Such a problem definition anticipates absolute minimal cost solutions as the goal, and is not necessarily appropriate for applications in which routing time is important. In such a case, a *good* solution is often better if it can be computed in significantly less time. This may be achieved by restricting the algorithm's search of the topology through heuristics which use additional information about the environment to avoid useless avenues of exploration.

This paper defines some additional characteristics of computer communications networks that expand the information over that provided in the original problem statement, and then uses this information in the application of heuristics to the routing problem. It has been argued[11] that only non-adaptive (or semi-adaptive) routing will be effective in the future environment of very large networks due to the enormous overhead that would be required for a fully adaptive approach. Non-adaptive routing is traditionally associated with centralized routing; therefore this paper addresses only this type. The results are from a study of the topology of real and simulated computer communication networks and the final topology of their routed paths.

Definitions

Long-Haul Network: Computer communications networks in which the physical distribution of resources and users may span large geographical areas, such as the entire United States. This paper is primarily concerned with large long-haul networks (e.g., $N > 500$) in which centralized routing suffers from computational complexity.

Link: A bidirectional communication path in a computer network with distinct ends.

Node: A terminus of network links, usually a traffic switching center, a computation center, or both.

Chain: A finite sequence of two or more links in which each pair of links within the sequence is connected by a single node of degree two.

Link cost: A real number $c(l)$ that is assigned to each link l in a computer network, periodically adjusted in value to control the distribution of traffic. Different costs may be assigned to each direction associated with a link for total generality; the results of this work may be applied in either case.

Connectivity (or degree) of a node: The number of links emanating from it to connect it with other nodes.

Characteristics of Long-Haul Networks

Large computer communication networks possess several very important characteristics that can be used to develop heuristics to accelerate the routing algorithms. This section represents a study of the inherent characteristics of the TYMNET computer network as it has matured through three developmental stages. These characteristics are representative of most long-haul networks, and the derived heuristics are universally applicable.

Geographical

When a long-haul network is relatively small (e.g., $N < 50$), most of its nodes will be found in large cities. Using the United States as an example, nodes would be found near population centers along the east and west coasts, in the south, and in the north-central area (e.g., Chicago), with long communication lines spanning the large central region. These first nodes serve as initial computational centers, around which large local networks develop as the network expands. As computational loads increase, more nodes are added in other cities. Once the network reaches about 200 nodes, the nodes will be fairly evenly distributed across all the states.

Connectivity

Distributions of connectivity for three different stages of the development of the TYMNET computer network are shown in Figure 1. Small networks are relatively loosely connected, and network growth is followed by a limited growth of average connectivity. Correspondingly, nodes of degree one were predominant when TYMNET was still small; as the network expanded, nodes of degree two became predominant. This growth is reflected by average connectivities of 2.4, 3.4, and 3.9 for 1971, 1975, and 1977, respectively. Taken together, nodes of degree one and two now comprise about one third of the total number of nodes.

Link costs

Network performance (e.g., response time) is very sensitive to traffic distribution that is directly controlled by the assignment of link costs (or lengths, as defined in Dijkstra's problem statement). In particular, the routing algorithm and cost assignment should focus on the following objectives:

1. Avoid congestion and balance the load across the network.
2. Minimize the response time or delay for interactive users.
3. Provide adequate bandwidth for high-speed users.
4. Maximize the resource utilization.

The actual assignment of link costs in TYMNET,[5] shown in Table I, was used in this study as a basis on which to synthesize a model of the TYMNET network, with variations in the total number of nodes and average connectivity.

Time variance

Computer communication networks are time-varying systems, and as such their parameters must be occasionally adjusted to accurately reflect changing conditions. For example, the assignment of link costs is adjusted periodically to control the distribution of traffic within the network, typically according to the time of day or day of the week. When the network is heavily loaded, utilization of individual nodes and links can change quickly; link costs must then be more frequently reassigned in order to maintain an equitable distribution of the workload, and thus a reasonable level of performance. For example, in the TYMNET network, the costs

TABLE I. Link Costs in TYMNET[5]

	Cost (arbitrary units)		
Line Type	Line not Overloaded	Overloaded One Way	Overloaded Both Ways
9600 bps	10	26	42
7200 bps	11	27	43
4800 bps	12	28	44
2400 bps	16	32	48

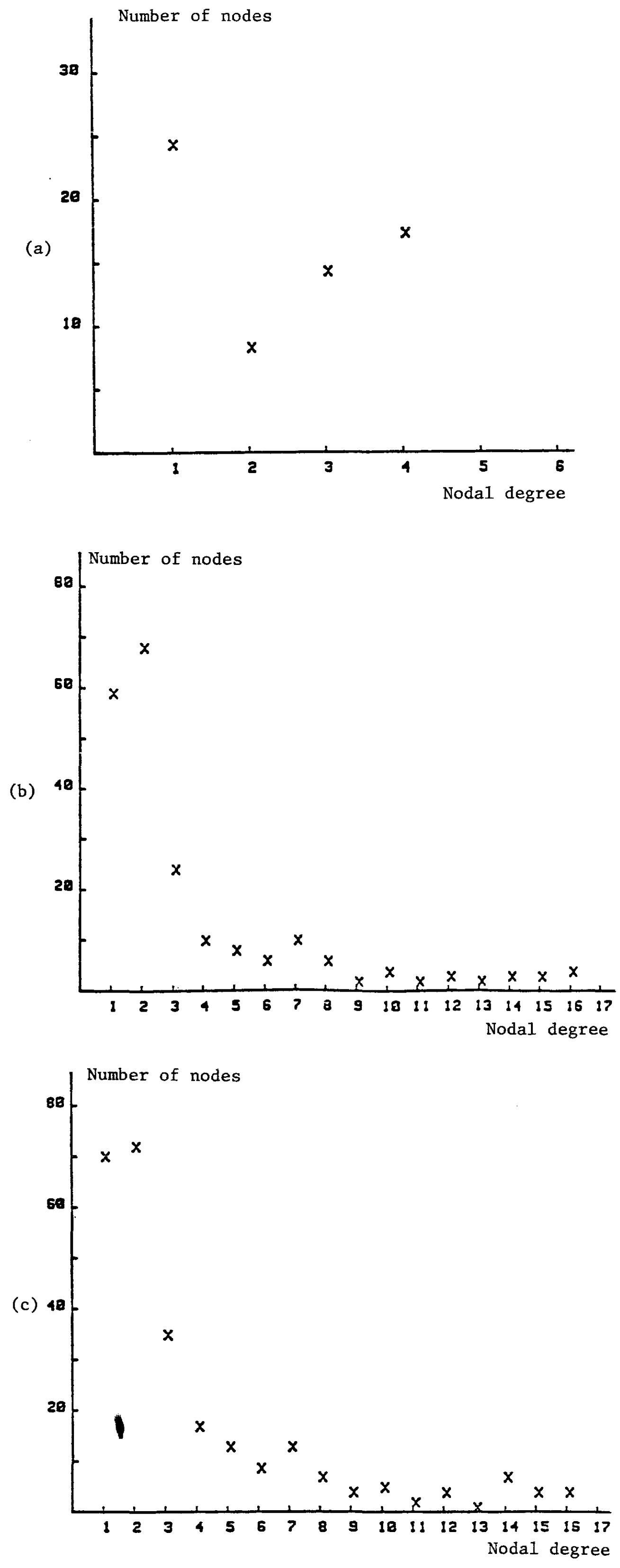


Figure 1—Connectivity distribution for the TYMNET network (a) 1971, (b) 1975, and (c) 1977

are usually reassigned once every few minutes, but sometimes as often as every few seconds during periods of high load, such as around noon.

Actual network topology changes less frequently, usually owing to the addition or deletion of nodes and links as part of normal growth. A typical network can grow as fast as a node per week. Long-term failure of a node or link may, of course, require a change in topology, but more often this is handled as a temporary assignment of infinity cost (no connection) to the link (or links) involved.

Network Synthesis for Simulation

For the construction of a network model for simulation purposes, it can be safely assumed that small networks have average connectivities in the range 2 to 3, medium networks in the range 3 to 4, and large networks greater than 4. In order to evaluate the new routing heuristics, a computer program was written to model long-haul networks with varied characteristics. The program consists of four parts:

1. Part one is responsible for placing a desired number of nodes N over a rectangular region representing the United States. The nodal coordinates are generated using a random number generator with a uniform probability, and a period longer than 50,000 numbers.
2. After nodal placement, the second part of the program assigns a connectivity degree to each node. The user supplies the desired average connectivity and maximal allowed nodal degree for the network to be constructed. The program first computes the number of links to be placed using the formula

   number of links L = (number of nodes $*$ average connectivity)/2,

   and then solves a system of equations:

   $$\sum_{i=1}^{i_{\max}} N_i = N, \tag{1}$$

   where N_i is the number of nodes of degree i,

   $$\left[\sum_{i=1}^{i_{\max}} N_i * i\right] \Big/ 2 = L, \tag{2}$$

   $$N_i = \left[c_1 \exp\left(-(i-2)^2/c_2^2\right)\right], \tag{3}$$

   where $200 < N \leq 300$, $i_{\max} = 9$; $100 < N \leq 200$, $i_{\max} = 6$; $0 < N \leq 100$, $i_{\max} = 3$;

   $$N_k = \left[N - \sum_{i=1}^{i_{\max}} N_i\right] \Big/ (\text{max nodal degree} - i_{\max}), \tag{4}$$

   max nodal degree $\geq k \geq i_{\max}$. Equation (3) models the distribution of nodes of small degree (exponentially, with constants $c1$ and $c2$ supplied by the user) while equation (4) models the distribution of nodes of large degree. The $N(i)$ versus i function obtained by solving equations (1)–(4) is then used to assign a degree to each node. When a node without degree is found, the random-number generator provides an i and from the $N(i)$ curve a degree is assigned. It can be seen that this process prefers to assign lower degrees.
3. Link placement is done in two phases. The first phase connects all the nodes of degree one. Since the relative geographical position of each node is known, each node is connected to its nearest neighbors. This procedure is supported by the fact that most of these single-link nodes represent small users (with relatively small computational power) whose main goal is to enter the network at its nearest connecting point. The second phase places the remaining links. The placement is done according to the following algorithm: A node is picked at random. Then, another node is picked at random, and its distance d is calculated. This distance is compared against a distance $d1$ produced by an exponential function of the type

   $$d1 = \sqrt{5}\, e^{-x/k}, \quad x \geq 0,$$

   where x is generated by the random-number generator and the constant k is supplied by the user. If $d > d1$, the link is placed; otherwise another node will be selected. Note that once the node is connected by a number of links equal to its connectivity degree, it is removed from the pool of available nodes. Note also that the largest distance is $\sqrt{5}$, a diagonal of the 2 by 1 polygon that we chose to model the U.S.

   After all links are placed, a depth-first search algorithm is run to ensure the network's connectivity. If the network is found to be disconnected, the whole procedure is repeated. An example of a network generated by this program with 100 nodes is shown in Figure 2.
4. Finally, to each link a cost figure is assigned. The program is using Table I for its pricing policy. Generally, links connecting smaller nodes are assumed to be of lower baud rate, while large nodes are always connected by high-speed lines. All the networks are assumed to be medium loaded, giving equal selection chances to all three line-loading cases given in Table I.

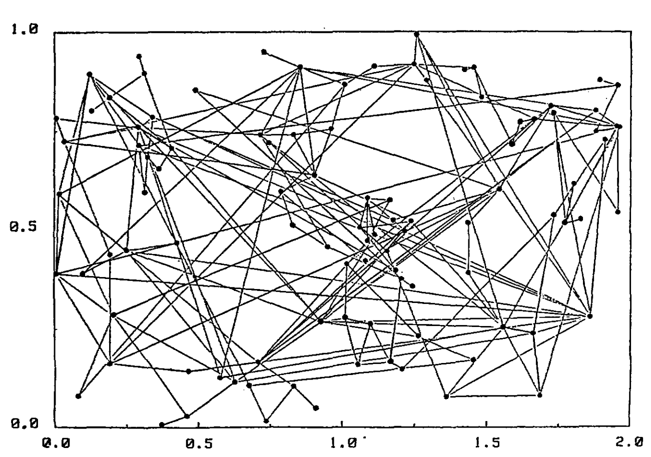


Figure 2—Plots of synthesized 30, 50, and 100-node networks

NETWORK REPRESENTATION HEURISTICS

This section describes the class of techniques that compress the representation of the network topology, and that require no changes to the actual routing algorithm. A small amount of work is required before each routing in order to initialize the topology representation. These techniques are not heuristics, because the final routed paths are identical to those produced without topology compression.

Nodes of degree one or two usually contribute very little to the topology of the final path found by a routing algorithm. For example, no more than two nodes of degree one will ever be in the final path; similarly, nodes of degree two can be thought of as unnecessarily splitting a link into two pieces. A large portion of the nodes in a typical network are of degree one or two, suggesting that these nodes be removed from the main representation of the network and temporarily reinserted on an individual basis, as required, before routing. This approach can be applied to any type of network that possesses similar characteristics. The shortest-path problem is then redefined as: We consider a network of N nodes, some or all pairs of which are connected by links; the cost of each link is given. We restrict ourselves to the case where at least one path exists between any two nodes. Once each time the network topology changes, the network representation must be preprocessed into three disjoint sets of nodes according to their degree.

Preprocessing

1. *Isolated Node Removal:* Nodes originally of degree one are removed along with their connecting link. The information necessary to reattach them later, individually, is saved in a separate data structure.
2. *Chain Collapsing:* All nodes originally of degree two are removed; the two links associated with each are replaced by a single link whose cost is the sum of the original links. The information necessary to reattach them later, individually, is saved in a separate data structure.

Routing

3. *Topology Initialization:* Replaces only those removed nodes (and associated links) that correspond to terminal nodes for the desired path.
4. *Minimal-Cost Path:* Finds the path of minimum total cost between the two given terminal nodes by applying any of the classical routing algorithms.

RESULTS

A classical Dijkstra's algorithm[5] was used as a standard for measuring the performance improvement due to restructuring the network representation as described above. Software counters measured the number of references made by the algorithm to data structures that represent the topology and link costs of the network, first for the original network L, and then for the reduced network L'. The overhead needed to reconstruct the network, if any of the terminal nodes happened to be of degree one or two, was included in a count obtained during the routing. The results were averaged over 1,000 node pairs selected at random as the terminal nodes to be routed. Modeled networks with 15, 50, 100, 150, 200, 250, and 300 nodes and average connectivities of 2.5, 3, 4, and 5 were considered. Average routing time improvements, measured as the computation ratio $M(L')/M(L)$ of memory references are plotted in Figure 3.

This preprocessing technique showed a clear improvement, saving on the average 15 to 35% of the memory references. The actual saving is very dependent on the particular network's topology as seen in Figure 3. Notice that the improvement is best for the networks containing smaller numbers of nodes; such networks inherently have smaller values of average connectivity, implying a higher percentage of nodes of degree one or two.

ROUTING ALGORITHM HEURISTICS

This section describes the class of techniques that require major modifications of the routing algorithm. These techniques are truly heuristics in the sense that the final routed paths are considered *good* instead of *best*; that is, the paths may have a slightly higher cost than those found by the classical algorithms, but offer significant savings in routing time.

COST-RESTRICTED ROUTING

Inherent to minimal-cost routing is the fact that a majority of the paths go through the lower-cost links. A high cost indicates an overloaded link; thus the desirability of using it is very small. This suggests that the topology could be preprocessed each time the link costs are reassigned in order to remove the very high cost links from the database. The shortest-path problem is then defined as: We consider a network of N nodes, some or all pairs of which are connected by links; the cost of each link is given. We restrict ourselves to the case where at least one path exists between any two nodes. Each time the link costs are reassigned, the network representation must be preprocessed into two disjoint sets of links according to their associated costs.

Preprocessing

1. *Costly Link Removal:* Reduce the degree of those nodes whose degree is greater than a specified value D by removing a specified number d of their most costly links.

Routing

2. *Minimal-Cost Routing:* Try to find the path of least total cost between the two designated terminal nodes via the remaining links by applying any of the classical routing algorithms.

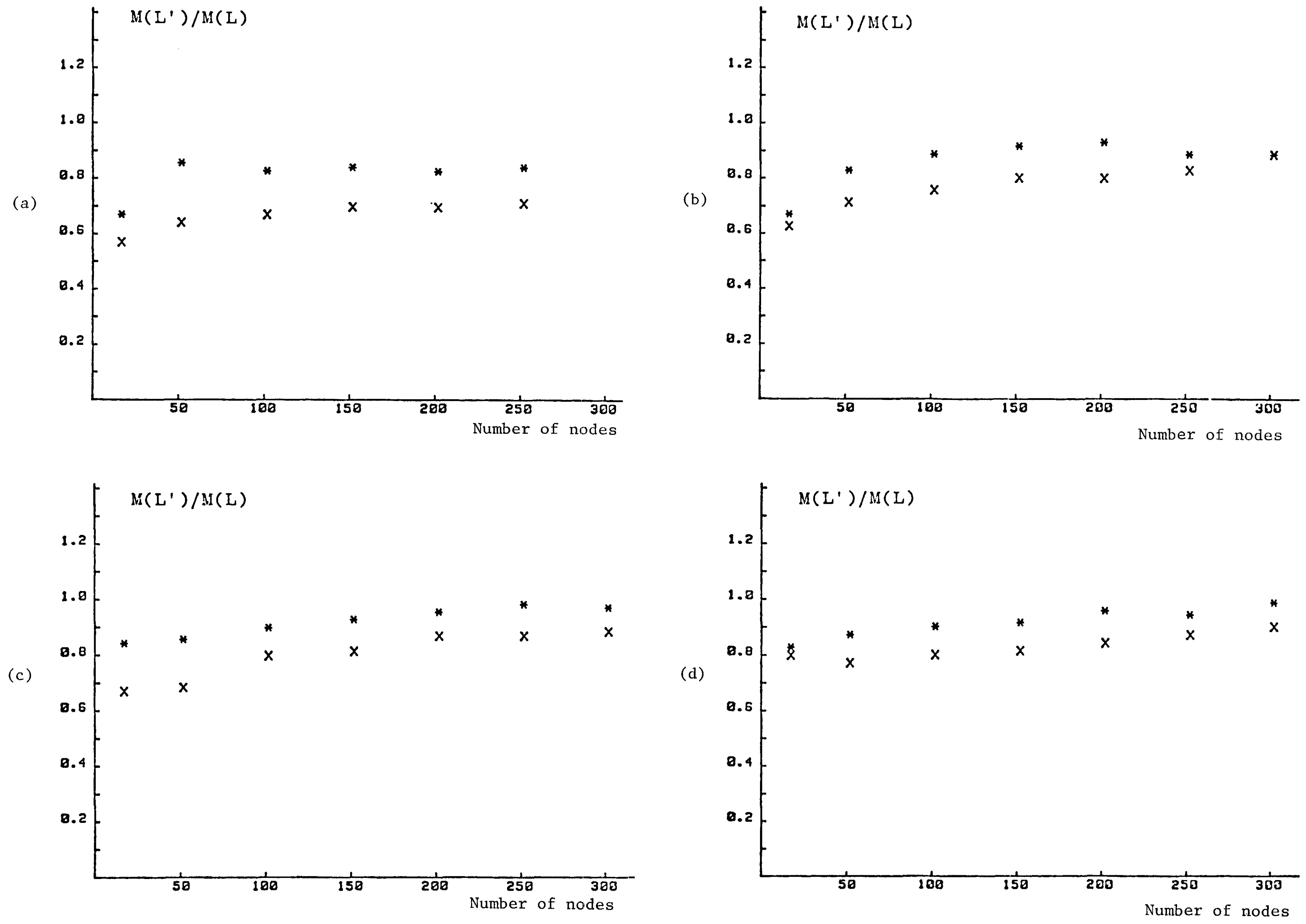


Figure 3—Computation ratio $M(L')/M(L)$ versus number of nodes N for various values of average connectivity. The upper set of points in each plot correspond to isolated node removal only; the lower sets correspond to combined isolated node removal and chain collapsing. Average connectivity: (a) 2.5; (b) 3.0; (c) 4.0; (d) 5.0.

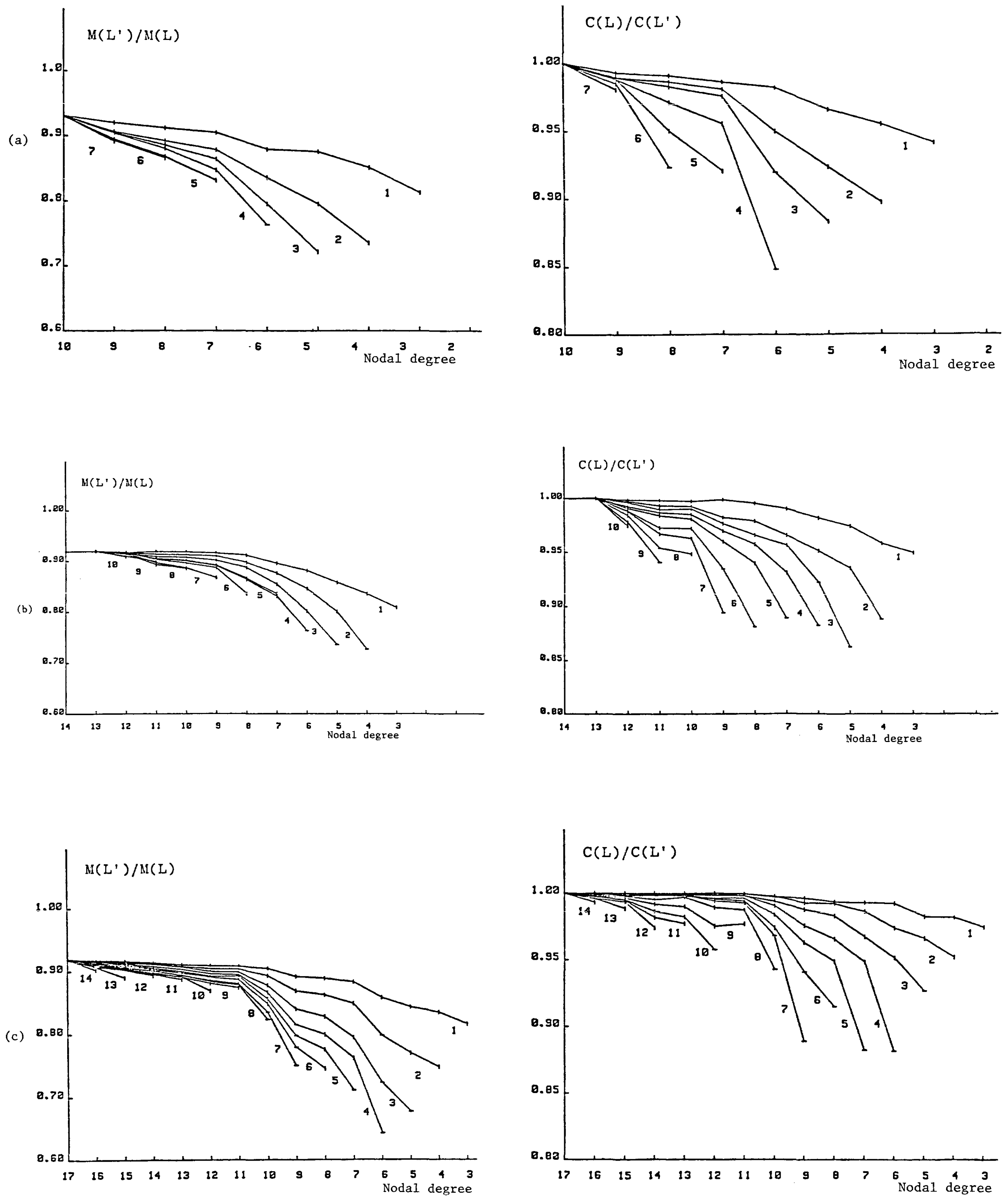


Figure 4—Cost restricted routing: Computation ratios $M(L')/M(L)$ and relative cost $C(L)/C(L')$ versus nodal degree for $N = 100$ (a), 200 (b), and 300 (c).

3. *Recovery Strategy:* If no path can be found, the routing algorithm is restarted using the original network representation. (Note: Other recovery strategies are of course possible, but our results do not justify a more complicated approach.)

Results

Long-haul networks of 100, 200, and 300 nodes were modeled based on TYMNET's cost assignment algorithm and nodal distribution; a typical average connectivity of 4.0 was used with a maximum allowed nodal degree of 16. A classical Dijkstra's algorithm[5] was used as a standard against which we measured the amount of routing computation and the relative cost of the final path. Software counted the number of references made by the algorithm to the data structures in memory that represent the topology and link costs of the network, first for the original network L, and then for the reduced network L'. The software also kept track of the total path cost found during the routing. The results were averaged over 1,000 node pairs selected at random as the terminal nodes to be routed. Average routing-time improvements, measured as the computation ratio $M(L')/M(L)$, and the relative cost of the final path, $C(L)/C(L')$, are plotted in Figure 4.

Three comments about Figure 4 are appropriate: (1) Each of the plotted $M(L')/M(L)$ curves corresponds to a particular value of links removed, d. For each such curve, the value plotted at nodal degree i corresponds to the removal of d links from all nodes of degree i or greater. (2) Since there were no nodes of degree 17 or greater, points at nodal degree of 17 represent the original unmodified network, L. The $M(L')/M(L)$ ratio is not 1.0; this is expected, since the links attached to each node were sorted before removal in the ascending order of their corresponding costs. The computation overhead necessary to perform the sorting operation is included in the total computation $M(L')$. As shown, this minor preprocessing alone is responsible for an 8% improvement. (3) The relative cost $C(L)/C(L')$ was nearly 1.0, until, in this removing procedure, the entire network consisted of nodes of degree five or less. This implied that the most important links attached to any node are the five cheapest.

For a particular network, a network designer needs to obtain a set of curves like the ones shown in Figure 4. From such set of curves, he can determine a set of values $d(n)$ of the number of links that can be removed from nodes of degree n while keeping the final path cost near that of the "best" solution, and providing the best overall routing time. Such an approach was used on our experimental network, producing the results shown in Figure 5. It shows that a 25 to 35% improvement can be achieved at a cost penalty of less than 10%.

DEGREE-RESTRICTED ROUTING

Figure 6a shows that the innermost links of TYMNET routed paths tend to pass through nodes of relatively high degree. For example, about 80% of an average path consists of nodes of degree six or greater; even if the threshold is raised from six to nine, such nodes still represent 50% of the nodes visited by typical routed paths.

This result may have been due to the fact that nodes of high degree in TYMNET are usually major switching centers in which the attached links offer high-speed (low assigned link

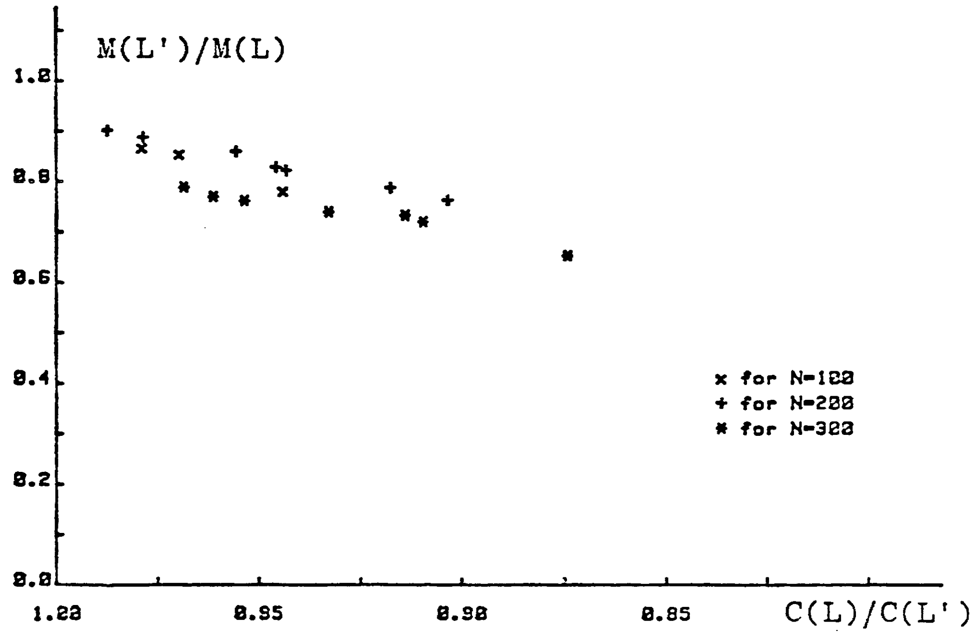


Figure 5—Cost restricted routing: Computation ratio $M(L')/M(L)$ versus relative cost $C(L)/C(L')$.

TABLE II. Distribution of nodal degree for the 300-node network used in the degree restricted routing experiment

Degree of a node	No. of nodes
1	45
2	64
3	25
4	40
5	26
6	21
7	16
8	13
9	8
10	6
11	10
12	5
13	2
14	7
15	4
16	8

cost) communications to other similar nodes. To explore this possibility, a second network was modeled that was identical in every respect to the first except that the link costs were assigned in a purely random manner. Specifically, the network had 300 nodes, link costs in the range of 1 to 45, and the nodal degree distribution shown in Table II. A thousand node pairs were selected at random as terminal nodes to be routed, with the results shown in Figure 6b. The marked similarity between Figures 6a and 6b suggests that the fact that paths pass through nodes of high degree has little to do with the costs of their attached links and more to do with the obvious fact that their greater connectivity is useful to the routing process.

In the 1977 distribution of TYMNET nodes shown in Figure 1c, fewer than 12% of the nodes—30 out of 251—are of degree six or greater. This suggests that a small subnetwork consisting of nodes of high degree could be extracted from the network to serve as a "backbone" for routing even though the network may never have been designed around a backbone. Since these large nodes constitute only a minor portion of the total, routing within this subnetwork can be accomplished very quickly.

Routing the end pieces from the terminal nodes to the subnetwork entry points is also very fast because these pieces contain very few links. For example, Figure 6 indicates that each end piece is usually less than 20% of the total path length. Measurements done on the 300-node network indicated an average path length of 4.6 nodes; 20% of this would then be less than one link.

The combination of these two routing steps is called *degree-restricted routing* rather than backbone routing because it is descriptive of the strategy, and so that the requirement of a network originally designed around a backbone is not implied. However, the adjective "backbone" will be used as a convenient means of identifying the extracted subnetwork consisting of nodes of high degree. The shortest-path problem is then defined as follows: We consider a network of N nodes, some or all pairs of which are connected by links; the cost of each link is given. We restrict ourselves to the case where at least one path exists between any two nodes. Each time the topology changes, the network representation must be preprocessed into two disjoint sets of nodes according to their degree.

Preprocessing

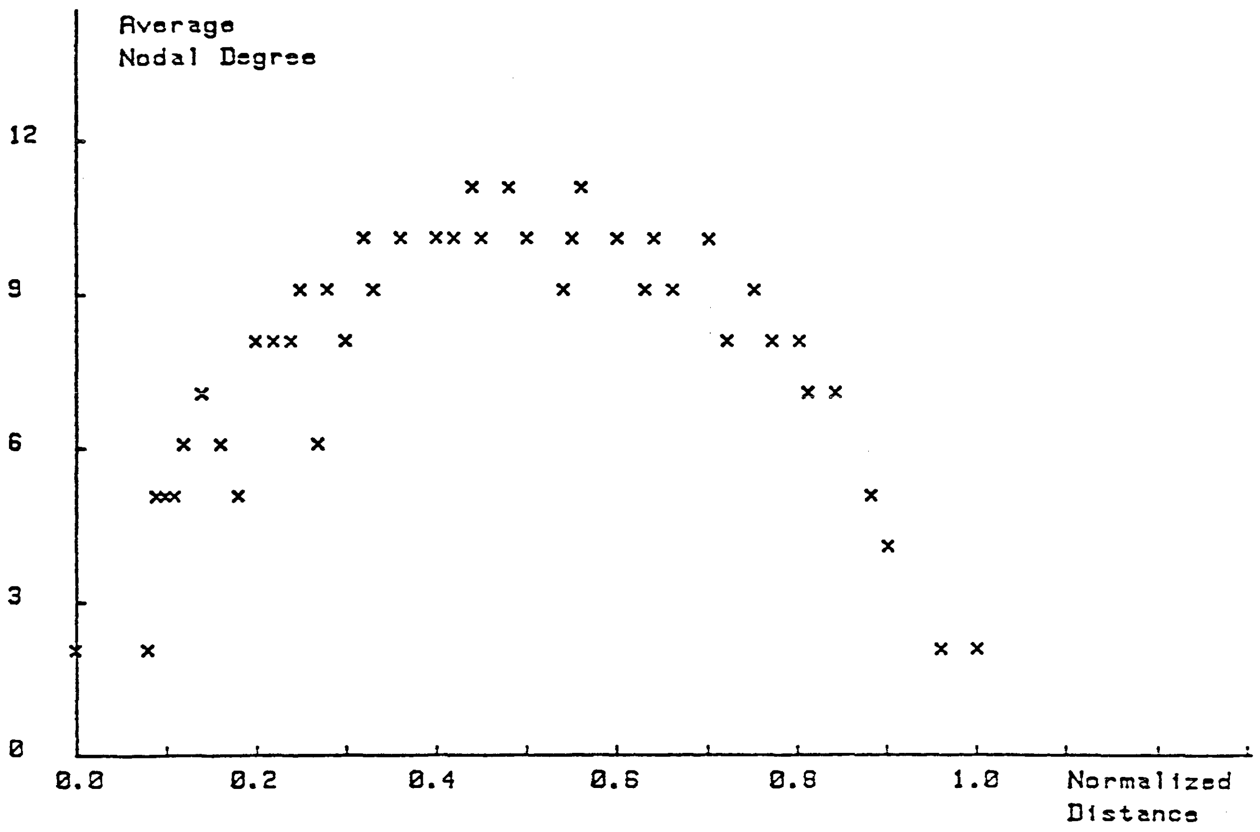
1. *Backbone Extraction:* A backbone subnetwork B consisting of all the nodes of a specified degree D or greater, together with those links which are connected to exactly two of these nodes, is extracted from the original network L.

Routing

2. *Part I (Terminal Nodes to Backbone Nodes):* For each of the terminal nodes, if it is a backbone node (degree of D or greater), provide it as a starting node for Part II without further computation. Otherwise, find all paths of a specified limited depth using minimal-cost routing from the terminal node into the network L. In so doing, try to collect a set of nodes of degree D or greater that are encountered along such paths, and provide this set of starting nodes to Part II. If no such nodes are found, go directly to step 4.
3. *Part II (Backbone Nodes to Backbone Nodes):* The two sets of large nodes obtained in Part I are to be connected by a minimal-cost path within the backbone subnetwork B. If these two sets share one or more nodes in common, there already exists at least one path between the terminal nodes; select the minimal-cost path from this set of path(s). If the two sets of large nodes have no node in common, then routing between these sets is done using any of the classical routing algorithms modified to find the lowest-cost path between several alternative sources and destinations, restricting the search to the backbone subnetwork.
4. *Recovery Strategy:* If no nodes of degree D or greater are found in Part I, the routing process is restarted using the original network L and any of the classical algorithms. (Note: Other recovery strategies are of course possible, but our results do not justify a more complicated approach.)

The maximum routing depth allowed in Part I must be sufficient to enter the backbone and collect a reasonable number of initial routing nodes for Part II. Figure 6 suggests that this depth need only be a couple of links; in our experiments, we used a conservative depth of three. It is actually desirable to have different values for D in Part I and Part II. In Part II, a value of D large enough to restrict the total size of the backbone subnetwork is needed; however, in Part I this value may be too large to reliably collect even a single entry to the backbone. Using different values of D in the two parts implies

(a)



(b)

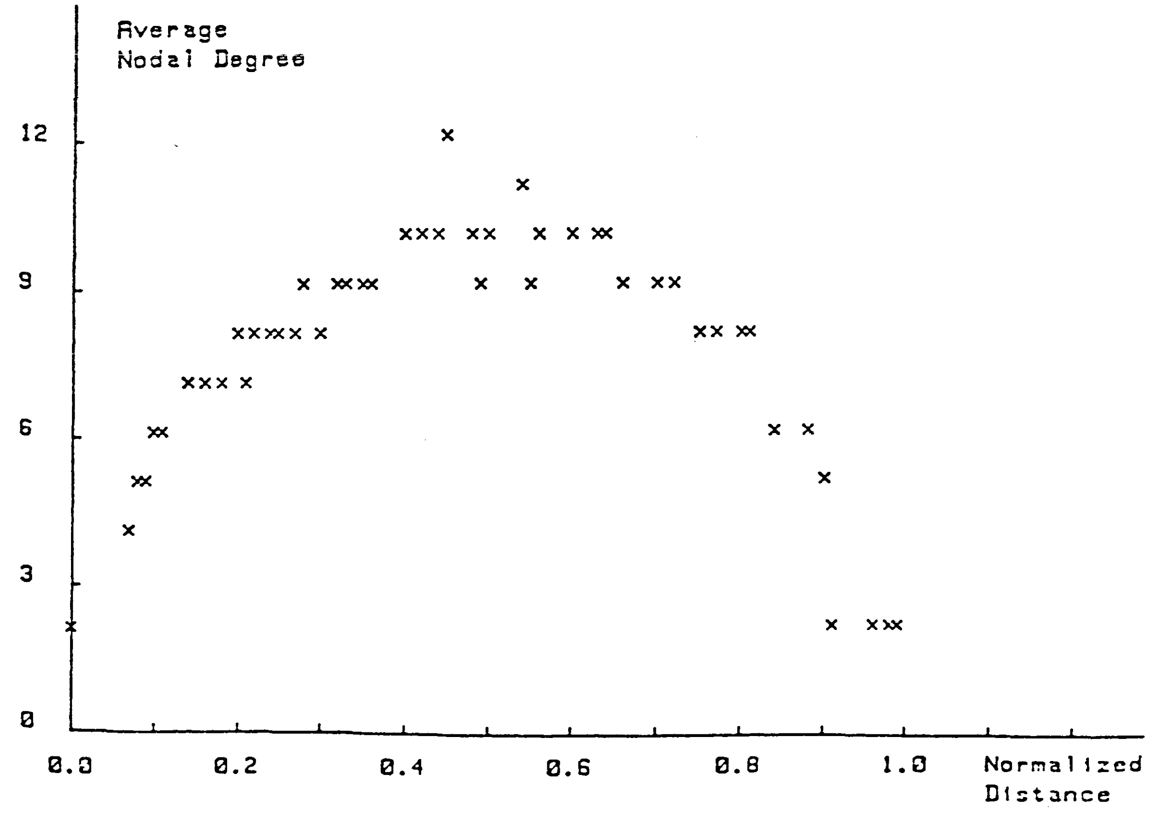


Figure 6—Average nodal degree as a function of normalized path length for 1,000 random node pairs routed with (a) TYMNET cost assignment, and (b) random cost assignment.

that the Part II must accept initial starting nodes whose degree may be smaller than its own subset of nodes, with the agreement that the recovery procedure must be invoked if none of the initial nodes can be routed to a backbone node with a single link. Trying to explore this extra level in Part I would require search of a much larger topology, and thus represent much more computation.

The routing algorithm developed first checked the nodal degree of each terminal node. If both nodes, according to their nodal degree, belonged to the backbone part B, their routing was done strictly through the set of the backbone nodes. If one node belonged to the backbone B and the other was outside B, the outside node is first routed as described in Part I; collected nodes are then routed with the second terminal node as described in Part II.

Lastly, if both nodes were positioned outside the backbone, one node is routed as described in Part I. Two different cases can arise: (1) if in this routing process the other terminal node is encountered, Part I will continue until its normal termination and then the minimal path will be chosen. In this special case the backbone nodes are not used for routing. It was noted that a significant number of nodes were routed like this, which is a consequence of the fact that most of communications are local. (2) If the other terminal node is not encountered during the routing of the first node, Part I is repeated with the second terminal node. After its completion, Part II is used as already described.

Results

In order to determine suitable values for D in Part II, several trial values of D were selected and used to extract a corresponding backbone subnetwork B from a modeled network L of 300 nodes. For each trial value, 1,000 pairs of backbone nodes were randomly selected and routed within L and then B. The average computation ratio $M(B)/M(L)$ and relative cost $C(L)/C(B)$ for various values of D are shown in Figure 7.

In order to determine a suitable value for D in Part I, several trial values of D were selected and used as a criterion for collecting backbone entry nodes. For each trial value of D, another 1,000 pairs of nodes, one from B and another from L-B were selected and routed. Average computation ratio $M(B)/M(L)$ and relative cost $C(L)/C(B)$ for various values of D are also shown in Figure 7.

The results confirm that different values of D should be used for the two parts of the algorithm. To minimize the errors, a value of 6 was chosen for Part II and 5 for Part I. After these values had been selected, another 1,000 random pairs of node pairs from L were routed using the degree-restricted routing algorithm defined above and compared to routing using the classical algorithm. Note that if the recovery strategy is used, the computation complexity is determined by summing the total number of references made by the failed routing process and the number of references made during the recovery procedure. The overall improvement in memory references was 48.52% with only a 2.32% relative increase in cost of the final path.

SUMMARY

This paper described three new approaches to the problem of finding low-cost paths in computer communication networks. Further information on the details of the actual experiments may be found elsewhere.[12] The simulation results have demonstrated that algorithms based on these ideas produce excellent performance at nearly minimal cost for medium and large networks. Since the algorithms use only the network's topology and link costs (without necessarily knowing that the graph represents a computer network), similar results can be achieved for other networks that possess similar topological properties, and such heuristic solutions will be even more important when these same networks grow beyond 1,000 nodes.

The domain of the routing algorithm can be compressed by unconditionally removing nodes of degree one and two, producing a simpler network of fewer nodes and higher average connectivity. This technique provides a considerable improvement in speed with no increase in the cost of the final path; it is applicable to any type of network. Removing this small set of nodes (about a third of the total) resulted in a significant reduction of the computation required for the routing, since the required computation is proportional to the square of N.

Cost-restricted routing yields improvements dependent on the ratio of eliminated links to the total number of links, but it also may result in a small increase in the final path cost. The choice of a particular threshold or thresholds of cost for ignoring links will depend on the network, and should be determined from curves similar to those shown in Figure 4. Note that the results presented did not include a computation overhead for restructuring the database after the loading situation throughout the network is significantly changed. A separate experiment was run to include this overhead; 3% of the improvement is lost if the restructuring is done after every 20 routings. This drops to below 1% when the restructuring is done after every 50 routings. In practice, the loss will be somewhere in between, depending on the part of the day the network is used and on its loading.

Degree-restricted routing, like cost-restricted routing, requires preprocessing. However, experiments have shown that degree-restricted routing is more sensitive to connectivity than cost assignment is; thus the backbone extraction need only be performed when a link or node is actually removed or added. Moreover, the selection of value(s) for D reflects average characteristics of the network's topology, and is fairly insensitive to minor topology changes; thus the value(s) of D may be recalculated very infrequently (weeks for example).

As a final experiment, a combination of the node removal and degree-restricted routings was applied to the same 300-node network. The average saving in memory references was 54.72% with an average increase of 1.21% in the cost of the final path. This is an additional 6.20% improvement over the results obtained when only the degree-restricted routing algorithm was used. The distribution of number of performed routings versus achieved improvement is given in Figure 8. The values for D were the same as in the experiment pictured in Figure 7. When 1,000 node pairs were routed, in only 8

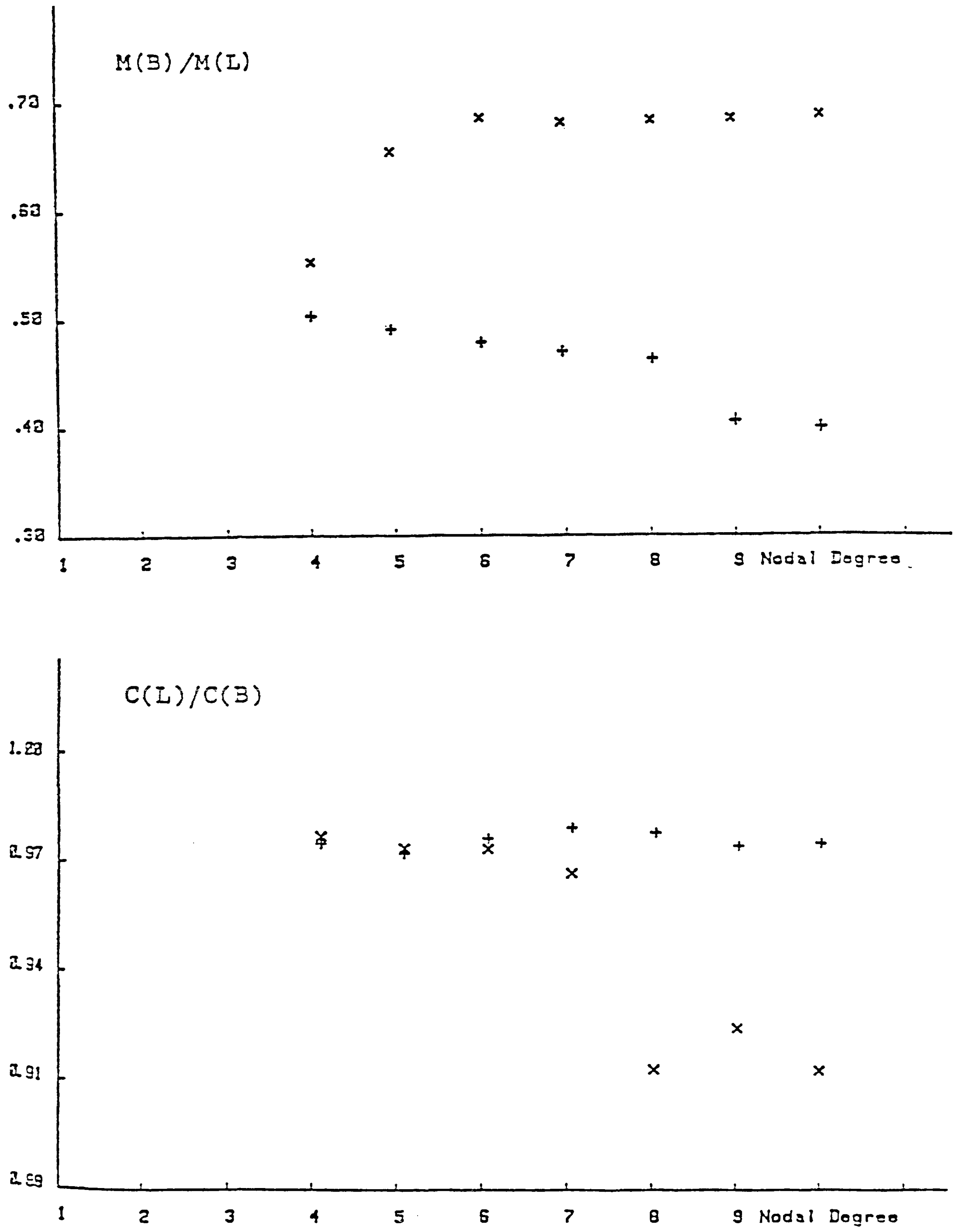
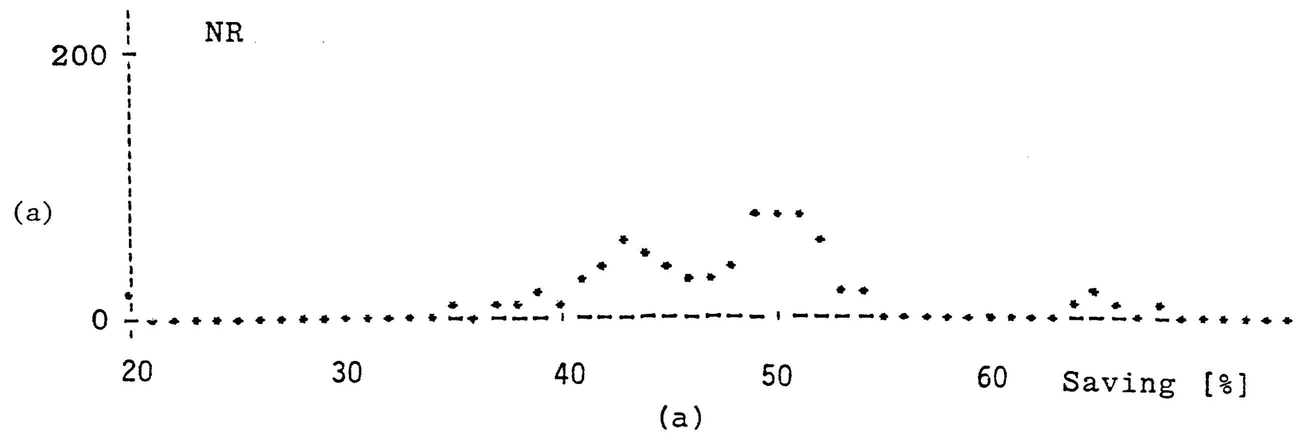


Figure 7—Degree restricted routing: Average computation ratio $M(B)/M(L)$ and relative cost $C(L)/C(B)$ versus nodal degree used as a breakpoint between large and small nodes. The points marked with an " × " represent large node to large node routing; those marked by a "+" represent large node to small node routing.



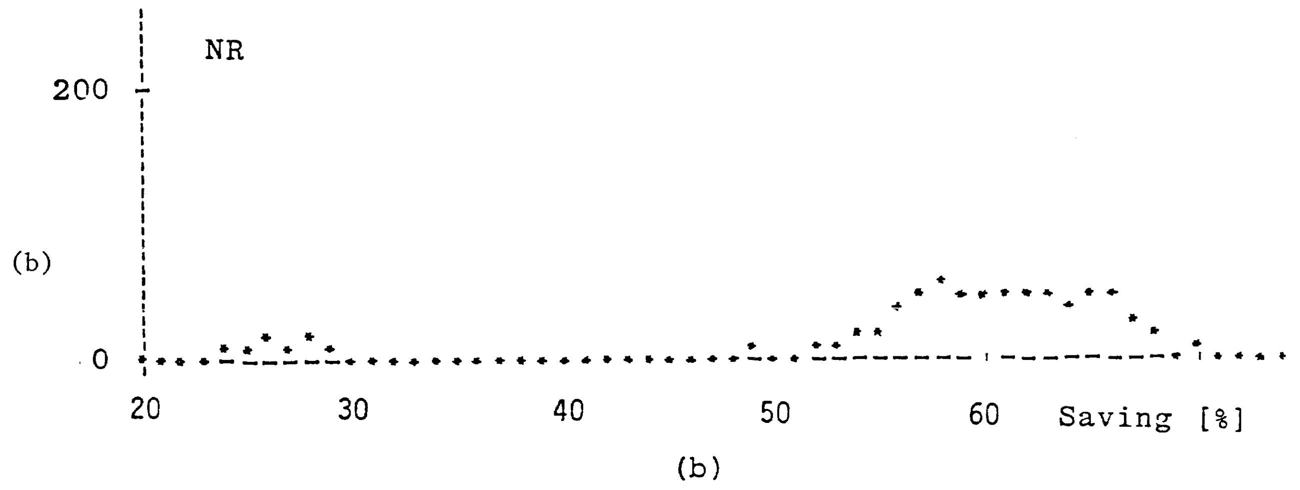


Figure 8—Distribution of *NR* (number of routings) versus achieved Saving (given in %), when degree restricted routing algorithm was applied (a) on unmodified 300-node network, and (b) on the same network but with nodes of degree one and two removed.

cases did the combined algorithm fail to find a path and have to use the help of the recovery strategy.

ACKNOWLEDGMENTS

The authors are indebted to Mr. J. Rindi and Mr. A. Rajaraman, both from TYMSHARE Corporation, for long and constructive discussions, and to the University of Santa Clara for their generous policy of free and unlimited computer time for students and faculty.

REFERENCES

1. Kruskal, J. B. Jr. "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem." *Proceedings of the American Mathematical Society*, 7 (1956), pp. 48–50.
2. Dijkstra, E. W. "A Note on Two Problems in Connexion with Graphs." *Numerische Mathematic*, 1 (1959), pp. 269–271.
3. Floyd, R. W. "Algorithm 97: Shortest Path." *Communications of the ACM*, 5 (1962), p. 345.
4. Yen, Jin-Yu. "A Shortest Path Algorithm," Ph.D. Dissertation, University of California, Berkeley, 1970.
5. Rajaraman, A. "Routing in Tymnet." *European Computer Conference '78*, London, May 9–12, 1978. Paper available from TYMSHARE Corporation.
6. Dantzig, G. B. "The Shortest Route Problem." *Operations Research*, 5 (1961), pp. 270–273.
7. Dantzig, G. B., W. O. Blatner, and M. R. Rao. "All Shortest Routes in a Graph," Technical Report 66-3, Operations Research House, Stanford University, Nov. 1966.
8. Yen, Jin-Yu. "Shortest Path Network Problems," unpublished monograph, 1971.
9. Kleinrock, L., and F. Kamoun. "Hierarchical Routing for Large Networks." *Computer Networks*, 1 (1977), pp. 155–174.
10. Gomberg, G. R. A., et al. "A Design Study of a Hierarchically Connected Packet-Switching Network Using Simulation Techniques." *Computer Networks*, 3 (1979), pp. 114–135.
11. Rinde, J. "TYMNET I: An Alternative to Packet Switching Technology." *Proceedings of the 3rd ICCC*, Toronto, 1976, pp. 268–273.
12. Pesic, I. M. "Some Heuristics for Centralized Routing in Large Long-Haul Networks," Ph.D. Dissertation, University of Santa Clara, January 1982.
13. Pape, U. "Algorithm 562: Shortest Path Lengths." *ACM Transactions on Mathematical Software*, 6 (1980), pp. 450–455.
14. Schwartz, M., and T. E. Stern. "Routing Techniques Used in Computer Communication Networks." *IEEE Transactions on Communications*, COM-28 (1980), pp. 539–552.

A new probabilistic routing algorithm for packet-switched computer networks

by CHI-YUAN CHIN and KAI HWANG
Purdue University
West Lafayette, Indiana

ABSTRACT

A probabilistic method is proposed for message routing in packet-switched computer networks with distributed control. The routing table associated with each node consists of path entries, instead of branch entries as found in most routing schemes. Packets are assigned with different paths on a probabilistic basis. The path selection is entirely processed at the source node. The routing table is updated dynamically with change of packet-generating rates at all nodes. We introduce a new quantitative measure, *path capacity,* to model each path as an $M/M/1$ queue. With the path capacities, routing tables are updated frequently to achieve balanced minimum delays among all paths. The update overhead is a constant, independent of the size of the network. Both analytical and simulation results are presented and compared with the new ARPANET routing method under various traffic conditions. This probabilistic, path-directed routing algorithm performs significantly better than the new ARPANET routing method under moderate and heavy traffic conditions. Under very light traffic conditions, the two methods have almost equal performance. This method can be applied to improve packet routing in any computer communications networks with distributed control.

1. INTRODUCTION

Existing routing schemes are all branch-directed in computer communications networks with distributed control.[4] By *branch-directed,* we mean that the routing decision of a packet is determined from node to node. Each node selects a branch to transmit the packet to its neighboring node. A *path-directed* routing method, on the contrary, predetermines the entire path of each packet at the source node. This method has been applied in centralized-control networks.[16] This paper proposes a probabilistic, path-directed method to simplify packet routing in distributed-control networks. Loop-free routing is enforced in this method. *Probabilistic routing* is often superior to deterministic routing in minimizing the average message delay.[13] In a probabilistic routing method, one out of multiple paths is chosen for routing a packet. In *deterministic routing* a unique path is assigned to route a packet to its destination.[2]

Different probabilistic routing methods have been suggested for achieving minimum delays. In Cantor and Gerla[3] packets are distributed among the branches by solving a nonlinear commodity flow problem. Gallager[7] proposed a minimum-delay routing algorithm by using a flow deviation and distributed computation method. Schwartz and Cheung[15] used a gradient projection algorithm to determine the routing paths. Because of long computational overhead, none of the above probabilistic routing methods can afford to update the routing tables frequently enough to catch up to the variation of network status in a real-life communications environment.

When the routing tables are periodically updated, the routing method is considered *adaptive.* Many adaptive routing methods have been developed.[18] The new routing algorithm for the ARPANET is adaptive, based on a shortest-distance-tree technique.[10, 14] Only one path between a source and a destination is selected at a time. In fact, it performs deterministically between two consecutive table updates. Boorstyn and Livne[1] proposed a two-level adaptive routing scheme based on a multiserver model. It is rather difficult to solve the multiserver model with multiple paths.

In this paper, we propose a *probabilistic and path-directed* (PPD) routing algorithm for a computer communications network with distributed control. The method is made adaptive by modeling each path as an $M/M/1$ queue. The service rate of each queue equals the *effective path capacity,* to be defined shortly. Each path is an entry in the routing table associated with the starting node. Paths to the same destination are grouped into a subtable. The use of each path is periodically checked and recorded in the subtables. A source node distributes packets among selected paths to achieve balanced and nearly minimum-delay performance. Each packet being transmitted is tagged with its path code to allow immediate routing at intermediate nodes.

In the next section we define traffic measures, path delays, and path capacities. In Section 3 we introduce a path encoding method, to save memory space and expedite the routing process, and then specify the PPD routing algorithm. The table update policies are described in Section 4. The performance of the PPD routing algorithm is analyzed in Section 5. Simulation results of the PPD method are given in Section 6, and compared with the new ARPANET algorithm. Finally two generalized ARPANET methods are proposed based on the proposed PPD routing algorithm.

2. NETWORK PARAMETERS AND MEASURES

Consider a network with n nodes denoted by $1, 2, \ldots, n$. Denote a branch from node i to node k by (i, k). We distinguish branch (i, k) from branch (k, i) to emphasize the traffic directions. A path from node i to node k via distinct intermediate nodes, $l, m, \ldots$, is represented by $(i, l, m, \ldots, k)$. No loop is allowed in a path. A path f contains a path g, denoted by $f \supset g$, if path g is a proper subpath of path f. For example, path (i, j, k) contains paths (i, j) and (j, k), but does not contain path (i, k) or path (i, j, k) itself. We use the symbol g to represent a general path $g \equiv (i, j, \ldots, k)$. The set of all possible paths from node i to node k is denoted by $\{i, k\}$. Furthermore, i and k denote a source node and a destination node, respectively in the sequel.

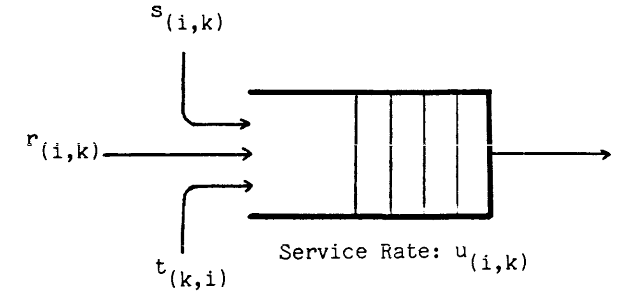
Basic notations are summarized in Table I. All measurements are expected values. The packet generation rate r_g is measured on the packets that are generated at node i, destined for node j, and routed via path g. All packets routed via path h where $h \supset g$ are recorded to measure the packet passing rate s_g. Both the generation rate $r_{i,k}$ and the passing rate $s_{i,k}$ are measured on all possible paths in $\{i,k\}$, i.e., $r_{i,k}=\sum_{g\epsilon\{i,k\}} r_g$, and $s_{i,k}=\sum_{g\epsilon\{i,k\}} s_g$. Note that $r_{(i,k)}$ and $s_{(i,k)}$ are subterms of $r_{i,k}$ and $s_{i,k}$, respectively, if branch (i, k) exists in a network. Furthermore, note that $\phi_g \geq 0$ and $\sum_{g\epsilon\{i,k\}} \phi_g=1$. Each packet, generated at any node, is preassigned with a path by examining assignment probabilities ϕ_g's of all paths in the set $\{i, k\}$. The branch capacity $C_{(i,k)}$ is an undirected quantity; i.e., $C_{(i,k)} = C_{(k,i)}$. Assume exponential message length with an average of l/L bits per packet. The packets generated at each node assume a Poisson distribution. If the traffic is light, the packet arrival rate at each node will not be affected by those at other nodes. Thus, the arrival rate at each node can also assume a Poisson distribution. Furthermore, each branch can be considered as an $M/M/1$ queueing model. Let $t_{(i,k)} = r_{(i,k)} + s_{(i,k)}$. According to a derivation in Klenrock,[9] the average packet delay $D_{(i,k)}$ on branch (i, k) is evaluated by the expression $D_{(i,k)} = 1/(L \cdot C_{(i,k)} - t_{(i,k)} - t_{(k,i)})$. Let $u_{(i,k)} = L \cdot C_{(i,k)}$, then we have

$$D_{(i,k)} = \frac{1}{u_{(i,k)} - t_{(i,k)} - t_{(k,i)}}. \qquad (1)$$

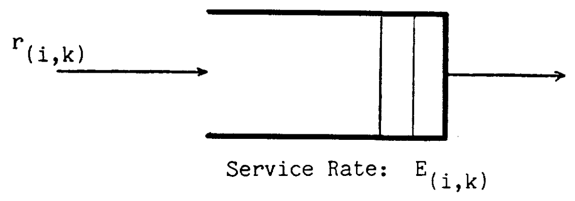
Figure 1(a) shows the $M/M/1$ queueing model of a branch (i, k) with capacity $u_{(i,k)}$. The arrival rate of the queue is composed of three components $r_{(i,k)}$, $s_{(i,k)}$, and $t_{(k,i)}$. In other words, the branch capacity $u_{(i,k)}$ is shared by three sources of arriving packets with expected arrival rates $r_{(i,k)}$, $s_{(i,k)}$, and $t_{(k,i)}$, respectively. Each packet source increases the load of branch (i, k); i.e., each source reduces the unused capacity of branch (i, k) by the quantity of its arrival rate. We now load the three packet sources on branch (i, k) separately as an example. After the packet sources with rates $s_{(i,k)}$ and $t_{(k,i)}$ have been loaded, the unused capacity of branch (i, k) reduces to $u_{(i,k)} - s_{(i,k)} - t_{(k,i)}$. We then load the last packet source, associated with rate $r_{(i,k)}$. The model now becomes a queue with arrival rate $r_{(i,k)}$ and service rate $u_{(i,k)} - s_{(i,k)} - t_{(k,i)}$, as shown in Figure 1(b). We define this service rate as the *effective capacity* of branch (i, k), associated with the packet source which has arrival rate $r_{(i,k)}$; i.e.,

TABLE I—Notations and definitions

Notation	Definition
$r_{i,k}$	The packet generation rate at source node i and destined for node k (associated with path set $\{i,k\}$) (packets/sec)
r_g	The packet generation rate at node i along a path $g=(i,j, \ldots, k)$ (packets/sec)
$s_g = \sum_f r_f) - r_g$	The packet passing rate at node i along a path $g=(i,j, \ldots, k)$, where f is a path containing the path g (packets/sec)
$s_{i,k} = \sum_{g \in \{i,k\}} s_g$	The packet passing rate at node i, which is destined for or passed through node k (packets/sec)
$\phi_g = r_g / r_{i,k}$	The assignment probability of path $g=(i,j, \ldots, k)$ in set $\{i,k\}$
$C_{(i,k)}$	The physical capacity of branch (i,k) (bits/sec)
$D_{(i,k)}$	The average branch delay per packet, transmitted from node i along branch (i,k) (sec/packet)
D_g	The average path delay per packet, transmitted from node i along path $g=(i,j, \ldots, k)$ (sec/packet)
$1/L$	The average packet length (bits/packet)
$E_{(i,k)} = C_{(i,k)} \cdot L - s_{(i,k)} - r_{(k,i)} - s_{(k,i)}$	The effective branch capacity of branch (i,k) at node i (packets/sec)
$E_g = r_g + \frac{1}{D_g}$	The effective path capacity of path $g=(i,j, \ldots, k)$ at node i (packets/sec)



(a) A branch queue.



$$E_{(i,k)} = E_{(i,k)}(r_{(i,k)}) = u_{(i,k)} - s_{(i,k)} - t_{(k,i)}$$

(b) A simplified branch queue.

Figure 1—Queueing models for a branch (i, k)

$$E_{(i,k)}(r_{(i,k)}) = u_{(i,k)} - s_{(i,k)} - t_{(k,i)} \qquad (2)$$

For short, we use $E_{(i,k)}$ to represent $E_{(i,k)}(r_{(i,k)})$. Since the model in Figure 1(b) is still an $M/M/1$ queue, the average packet delay of the packets with arrival rate $r_{(i,k)}$ can be evaluated by the following equation:

$$D_{(i,k)}(r_{(i,k)}) = \frac{1}{E_{(i,k)}(r_{(i,k)}) - r_{(i,k)}}. \qquad (3)$$

It can be shown that $D_{(i,k)}(r_{(i,k)})$ equals $D_{(i,k)}$, obtained by Equation 1. This means that we can use the queueing model of Figure 1(b) to correctly evaluate the average delay of the packets with arrival rate $r_{(i,k)}$. Equation 2 is useful for extending the concept from the effective branch capacity to the effective path capacity.

We introduce the concept of the effective path capacity by a simple example shown in Figure 2. Figure 2(a) describes a queueing model of path (i, j, k). Each branch of the path is considered as an $M/M/1$ queue. For discussion simplicity, we set $t_{(j,i)} = t_{(k,j)} = t_{(k,j,i)} = 0$. By Equation 2, the effective capacities of branch (i, j) and branch (j, k), associated with the packet source that has arrival rate $r_{(i,j,k)}$, are listed below:

$$E_{(i,j)}(r_{(i,j,k)}) = u_{(i,j)} - r_{(i,j)} - (s_{(i,j)} - r_{(i,j,k)})$$

$$E_{(j,k)}(r_{(i,j,k)}) = u_{(j,k)} - r_{(j,k)} - s_{(i,j,k)} - (s_{(j,k)} - s_{(i,j,k)} - r_{(i,j,k)})$$

$$= u_{(j,k)} - r_{(j,k)} - (s_{(j,k)} - r_{(i,j,k)})$$

With these two effective capacities, the queueing model in Figure 2(a) can be simplified as we did in Figure 1. The sim-

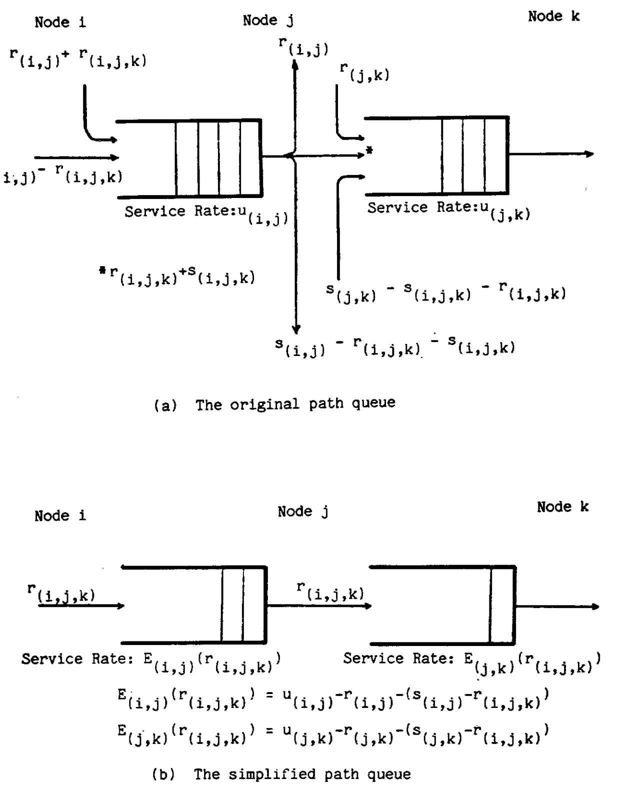

Node i
Node j
Node k
r(i,j)+ r(i,j,k)
r(i,j)
r(j,k)
s(i,j) - r(i,j,k)
Service Rate:u(i,j)
Service Rate:u(j,k)
*r(i,j,k)+s(i,j,k)
s(j,k) - s(i,j,k) - r(i,j,k)
s(i,j) - r(i,j,k) - s(i,j,k)
(a) The original path queue
Node i
Node j
Node k
r(i,j,k)
r(i,j,k)
Service Rate: E(i,j)(r(i,j,k))
Service Rate: E(j,k)(r(i,j,k))
E(i,j)(r(i,j,k)) = u(i,j)-r(i,j)-(s(i,j)-r(i,j,k))
E(j,k)(r(i,j,k)) = u(j,k)-r(j,k)-(s(j,k)-r(i,j,k))
(b) The simplified path queue

Figure 2—Queueing models for a path (i, j, k)

plified queueing model, shown in Figure 2(b), can be used to evaluate the average path delay $D_{(i,j,k)}$. In general, the average path delay D_g of a general path g can be derived from the model in Figure 3(a). It is composed of a sequence of queues with a unique arrival rate r_g. The service rate of each queue is the corresponding effective branch capacity, associated with the packet source that has arrival rate r_g. We now consider the queues as a path queue, shown in Figure 3(b). The path queue has the same arrival rate r_g and path delay D_g in Figure 3(a). Let the service rate of the path queue be the effective capacity of path g, denoted by E_g. The path queue can be approximated by an $M/M/1$ model as derived below; i.e., the average path delay D_g can be estimated by

$$D_g = \frac{1}{E_g - r_g}. \tag{4}$$

To calculate E_g from the effective branch capacities in Figure 3(a) is rather complicated. Directly using Equation 4 can avoid some time-consuming calculations. In other words, Equation 4 can be written as

$$E_g = r_g + \frac{1}{D_g} \tag{5}$$

The path delay D_g is measured by each message packet routed via path g. The delay measurement is sent back with the acknowledgment packet. After receiving the measurements, r_g and D_g are updated at source node i. By substituting r_g and D_g in Equation 5, we obtain E_g, which is used to predict D_g under different r_g subsequently. In other words, we can predict the delay performance of path g immediately after we reassign ϕ_g. It is helpful to achieve the best assignment of ϕ_g's by knowing the current path delay D_g's and traffic rates r_g's at source node i. This delay and rate is the most network status that we can have without interchanging information among nodes. If the quantity of the information is not enough (less than a threshold—e.g., 10 acknowledgment packets) to reflect the actual D_g, the delay information of all packets through path g will be involved to calculate D_g, and set $r_g + s_g$ as r_g.

In fact, the path queue is not exactly $M/M/1$. We now inspect the deviation of D_g, estimated by Equation 4, from the actual estimation by the model in Figure 3(a). By doing so, we can estimate the accurate working range of the $M/M/1$ pseudoqueue. Since there is a unique input r_g for each queue in Figure 3(a), arranging the order of these queues almost does not affect the estimation result D_g. Thus we may arrange the order of queues so that the service rate of each queue is greater than or equal to that of the queue on its left-hand side, and less than or equal to that on its right-hand side, as shown in Figure 4(a). In this way, only the first queue, which has service rate C_1, has packets waiting in it. It is thus modeled as $M/M/1$. The remaining $p-1$ queues can transmit the input packets without waiting. Since there is no waiting delay at these queues, the service delay at each of these queues is the inverse of the queue service rate. For example, the delay, required at the queue with service rate C_2, is $1/C_2$. We can therefore combine the last $p-1$ queues as a single queue with service rate C_b, where $C_b = 1/\left(\sum_{i=2}^{p} (1/C_i)\right)$ and p is the number of queues in Figure 4(a). Let $C_a = C_1$. Then we have an equivalent queueing model, as shown in Figure 4(b). A path queue of this queueing model is shown in Figure 4(c). Note that the path queue with service rate $C = E_g$ is considered an $M/M/1$ queue. The average path delays, estimated by the equivalent queueing model and the path queueing model, are denoted by $D_1(r_g)$ and $D_2(r_g)$ respectively, where r_g is the arrival rate. According to the properties of each model, $D_1(r_g)$ and $D_2(r_g)$ can be computed as follows:

$$D_1(r_g) = \frac{1}{C_1 - r_g} + \frac{1}{C_2}.$$

$$D_2(r_g) = \frac{1}{C - r_g}. \tag{6}$$

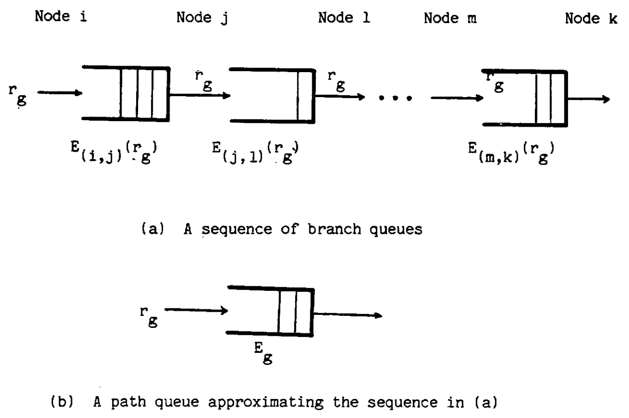
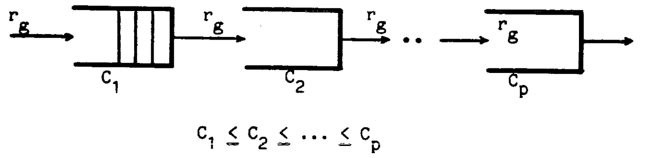
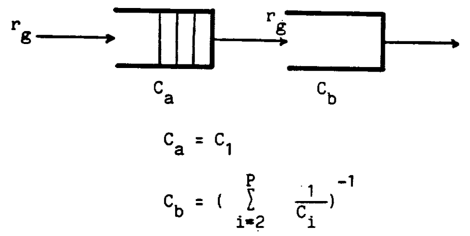

Node i
Node j
Node l
Node m
Node k
rg
rg
rg
...
rg
E(i,j)(rg)
E(j,l)(rg)
E(m,k)(rg)
(a) A sequence of branch queues
rg
Eg
(b) A path queue approximating the sequence in (a)

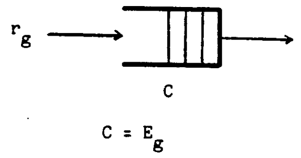
Figure 3—Queueing models for a general path $g = (i, j, l, \ldots, m, k)$ in a computer network



(a) A sequence of branch queues with increasing capacities.



(b) An equivalent path queue.



(c) A path queue approximating the queue in (b)

Figure 4—Modified queueing models for a path *g*

Assume that both models have the same estimation when $r_g = r$, i.e., $D_1(r) = D_2(r)$ and

$$\frac{1}{C_1 - r} + \frac{1}{C_2} = \frac{1}{C - r_g}. \tag{7}$$

We now inspect the difference between $D_1(r_g)$ and $D_2(r_g)$, when $r_g = r + h$. By Equation 6 and Equation 7, we have

$$\begin{aligned}\Delta &= D_2(r+h) - D_1(r+h)\\ &= \frac{1/(C-r)}{1-h/(C-r)} - \left[\frac{1/(C_1-r)}{1-h/(C_1-r)} + \frac{1}{C_2}\right]\\ &= h\cdot\left(2\cdot\frac{1}{C_1-r}\cdot\frac{1}{C_2} + \frac{1}{C_2}\right) + \frac{1}{h}\left[\left(\frac{h}{C_1-r} + \frac{h}{C_2}\right)^3\right.\\ &\qquad \left. - \left(\frac{h}{C_1-r}\right)^3\right] + \cdots.\end{aligned}$$

Since h is much smaller than $C_1 - r$ and C_2, we can neglect the higher-order terms in the above expression and obtain

$$\Delta = h\cdot\left(2\cdot\frac{1}{C_1-r}\cdot\frac{1}{C_2} + \frac{1}{C_2}\right). \tag{8}$$

By Equation 8, we may conclude that the delay difference Δ will be a relatively small quantity, if h is small. Furthermore, the assumption that the path queue is $M/M/1$ is accurate when the arrival rate r does not approach C_1. In other words, the assumption is accurate if a severe saturation condition does not occur in the network. Besides, the $M/M/1$ path queue overestimates the actual average path delay when the arrival rate increases (i.e., $h > 0$) and underestimates the delay when the rate decreases (i.e., $h < 0$). According to this property, a path will be supplied with fewer packets to avoid the occurrence of severe saturation cases in the network if the traffic rate on this path increases.

3. THE PPD ROUTING SCHEME

How to encode the paths is very important in saving memory space and expediting the routing process in the proposed PPD routing scheme. Our path-encoding scheme uses trunk numbers rather than node numbers. The reasons are two. First, the number of trunks is much smaller than the number of nodes in a large network. Second, the trunk number can be directly recognized by the network without any translation process to find the desired trunk.

The encoding scheme takes two steps at each node:

Step 1: Find every outgoing trunk number of the desired path from the source node to the destination node in order.

Step 2: Concatenate these numbers from right to left.

Figure 5 shows a sample network with trunk numbered. The path codes for path (1, 4, 8, 10) and path (5, 6, 7, 4, 9, 8, 11) are "1, 2, 3" and "2, 2, 3, 2, 1, 1," respectively. The digits in a path code can be in any kind of radix as long as the radix is greater than the largest trunk number in the network.

In the PPD routing scheme, each node has its own routing table. The table can be divided into $n - 1$ subtables, where n is the total number of nodes in the network. All possible paths to the same destination have entries in the same subtable. Let

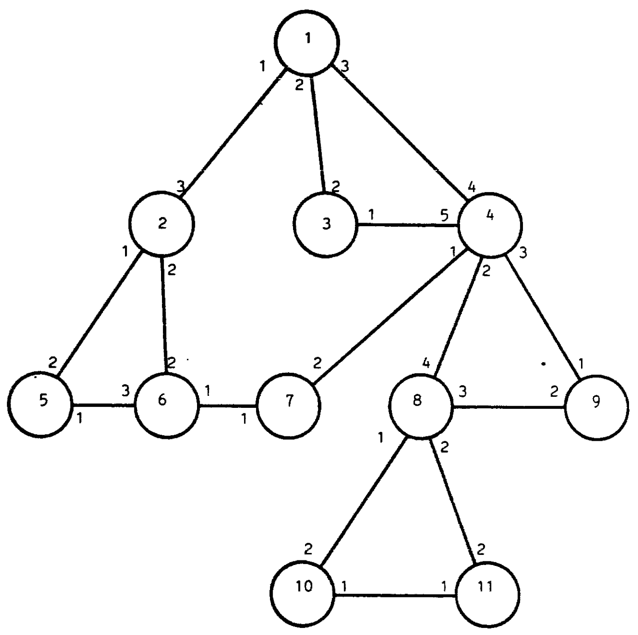


Figure 5—A network with branches labeled around each node

the subtable of node i, used for destination node k, be denoted by $U_{i,k}$. Each path entry of the subtable $U_{i,k}$ is a record structure containing the path code of path g, the path assignment probability ϕ_g, the measured generation rate r_g, and the measured average path delay D_g. Table II shows the subtables $U_{1,2}$ and $U_{1,3}$ of the network shown in Figure 5. If a network is too large to accommodate all same destination paths in a subtable, the number of paths will be limited. The limited number, say m, is a moderate number that depends on the available memory space. How to select m paths depends on the network structure. For simplicity, the m shortest paths, in the hardware sense, will be selected in this paper. The algorithm to find the m shortest paths for one destination can be found in Shier.[17] Since the shorter path g usually has larger ϕ_g, most of the message packets will be routed through the shorter path. It has been shown by simulation that properly selecting the value of m scarcely degrades the delay performance.

After a message packet destined for node k has been generated at node i, the source node probabilistically assigns a path code by looking at the path assignment probabilities ϕ'_g of the subtable $U_{i,k}$. The probabilistic distribution process can be implemented by either software methods or hardware methods.[6,11] The software method needs more time complexity to execute, whereas the hardware needs more memory space.

When a path code has been assigned, a packet can be routed in the network by the following PPD routing algorithm:

```
Input:   A message packet with a path code $X$.
Output:  Route the packet to its next host or destination
           node.
Steps:   1. Retrieve the code $X$ from the packet.
         2. If $X = 0$, stop; i.e., the destination node is
              found.
         3. Right shift out $X$ one trunk number, and save
              the new $X$ in the packet;
            $K$: = the shifted-out trunk number.
         4. Transmit the packet through the out-going
              trunk $K$ to the next host (node).
```

TABLE II—Routing subtables $U_{1,2}$ and $U_{1,3}$ at node 1 of the sample computer network in Figure 5

Path Code	Generation Rate	Path Delay	Path Capacity	Assignment Probability
$U_{1,2}$:				
1	$r_{(1,2)}$	$D_{(1,2)}$	$E_{(1,2)}$	$\phi_{(1,2)}$
2,1,1,1,2	$r_{(1,3,4,7,6,2)}$	$D_{(1,3,4,7,6,2)}$	$E_{(1,3,4,7,6,2)}$	$\phi_{(1,3,4,7,6,2)}$
2,1,1,3	$r_{(1,4,7,6,2)}$	$D_{(1,4,7,6,2)}$	$E_{(1,4,7,6,2)}$	$\phi_{(1,4,7,6,2)}$
2,3,1,1,1,2	$r_{(1,3,4,7,6,5,2)}$	$D_{(1,3,4,7,6,5,2)}$	$E_{(1,3,4,7,6,5,2)}$	$\phi_{(1,3,4,7,6,5,2)}$
2,3,1,1,3	$r_{(1,4,7,6,5,2)}$	$D_{(1,4,7,6,5,2)}$	$E_{(1,4,7,6,5,2)}$	$\phi_{(1,4,7,6,5,2)}$
$U_{1,3}$:				
2	$r_{(1,3)}$	$D_{(1,3)}$	$E_{(1,3)}$	$\phi_{(1,3)}$
5,3	$r_{(1,4,3)}$	$D_{(1,4,3)}$	$E_{(1,4,3)}$	$\phi_{(1,4,3)}$
5,2,1,2,1	$r_{(1,2,6,7,4,3)}$	$D_{(1,2,6,7,4,3)}$	$E_{(1,2,6,7,4,3)}$	$\phi_{(1,2,6,7,4,3)}$
5,2,1,1,1,1	$r_{(1,2,5,6,7,4,3)}$	$D_{(1,2,5,6,7,4,3)}$	$E_{(1,2,5,6,7,4,3)}$	$\phi_{(1,2,5,6,7,4,3)}$

In Step 4 the packet is sent to the waiting queue of the outgoing trunk K—a first-in-first-out principle. For example, to route a packet through path (1, 4, 8, 10) in Figure 5, we would proceed as follows. By the path-encoding scheme, the code of path (1, 4, 8, 10) is "1, 2, 3." The packet starts at node 1. After the first 3 steps of the routing algorithm, the code "1, 2" is loaded in the packet. The packet is then sent to node 4 through trunk 3. At node 4, the code in the packet is retrieved and shifted again. At this time, "1" is loaded in the packet, and the packet is then transmitted to node 8 through trunk 2. At node 8, the same procedure will be repeated again. Finally the destination node 10 accepts the packet after the code "0" is detected.

After receiving a packet, the destination node sends back an acknowledgment packet to the source node through the same routing path in the reverse direction. The code of the reverse path can be obtained by appending the incoming trunk numbers during the routing procedure. The acknowledgment packet has the highest priority to pass through the network. The source node can quickly receive it and record the accompanying path delay information. Since the length of an acknowledgment packet is very short, the traffic load is scarcely affected.

In the PPD routing scheme, the routing path of a packet is determined by its source node. Other intermediate nodes only pass the packet without making a path decision. In other words, no routing table is referred at intermediate nodes. Regardless of the waiting delay, the worst time complexity to route a packet is $O(s + (n-1) \cdot c)$. The worst time complexity to search in the routing table is s, and c is a constant time required for executing the PPD routing algorithm once. Note that the complexity of the same process in a branch-directed routing method is $O[(n-1) \cdot (s+c)]$. Usually a binary searching method is used for searching the routing tables. Then s becomes $O(\log n)$, and the worst time complexity of routing a packet in a branch-directed method is improved from $O(n \log n - \log n + c \cdot n - c) = O(n \log n)$ to $O(\log n + n \cdot c - c) = O(n)$ by the PPD method.

4. ROUTING TABLE UPDATE POLICIES

In the proposed routing table update policies, each node updates its routing table locally. At a node, different update processes for different subtables do not have to be executed at the same time. The steps to update the subtable $U_{i,k}$ of node i are listed below.

```
Step 1: Calculate every effective path capacity $E_g$ by Equa-
          tion 5, where $g \in \{i, k\}$.
Step 2: $E_{\max} = \max\{E_g, g \in \{i, k\}\}$.
Step 3: For any $E_g$, if $E_g/E_{\max} \geq A$, collect $E_g$ in a capacity
          set $S$, where $A$ is a threshold.
Step 4: $F_{i,k} = \sum_{E_g \in S} E_g$.
Step 5: $\phi_g = E_g/F_{i,k}$, for all $E_g \in S$;
        $\phi_g = 0$, for all $E_g \notin S$.
```

The worst time complexity of this update process is $O(m)$, where m is the maximum number of path entries in the subtable. In other words, the complexity $O(m)$ is a constant that is independent of the network size.

The update period for subtable $U_{i,k}$, denoted by $I_{i,k}$, is dependent on the present $r_{i,k}$. A greater $r_{i,k}$ helps to collect enough delay information within a shorter interval. Furthermore, a greater $r_{i,k}$ increases the traffic load on a network. By Equation 8, the heavier the traffic load, the more is required to improve the delay performance. Therefore, we define

$$I_{i,k}=\min\left\{\frac{B'}{r_{i,k}},\ B'\right\},$$

where B is a constant and B' is a threshold to avoid long $I_{i,k}$. It means that we need enough delay information but maintain the update ability. How to optimally set constants B and B' depends on the network structure.

Since the subtable $U_{i,k}$ is updated individually, the newly generated packets, only destined for node k, will be blocked during updating $U_{i,k}$. Because of the simplicity of the update process, the blocking delay is very short, i.e., $O(m)$. In this way, although the worst time complexity, required to update the entire routing table of a node, is $O(m\cdot n)$, the update process does not interfere with the routing process of almost all packets. In other words, the time complexity of the blocking delay is $O(0)$ in most cases. Therefore, the overhead of update table processes is negligible.

5. PERFORMANCE ANALYSIS

The PPD method has minimized the overhead to route a packet. It updates the subtables separately so that the routing delay caused by the update process is at most a small constant. The worst time complexity to route a packet from its source node to its destination node regardless of waiting delay (queuing time) is $O(m)+O(s)+O[(n-1)\cdot c]=O(n)$, which is the sum of the complexity of the update subtable process, the table-searching process, and the transition process. The constants required for the three processes are m, s, and c, discussed in Sections 3 and 4. $O(n)$ is the best possible time complexity to route a packet in an n-node network without considering waiting delay. Thus the waiting delay parameter, ranging from 0 to ∞, dominates the delay performance in the PPD method. Since the routing tables manipulate the waiting delay of each packet, the network performance is mainly dependent on the table update process, investigated below.

Since D_g is related to r_g, the best assignment of ϕ_g for achieving the minimum average delay should be a function of $r_{i,k}$. We now analyze how the value of ϕ_g, obtained by the proposed algorithm, differs from the best assignment by giving $r_{i,k}$ and all E_g, where $g\in\{i,k\}$. Assume no common branch between any two paths of $\{i,k\}$. Thus the total delay of the packets, associated with $r_{i,k}$, is calculated as follows:

$$D=\sum_{g\in\{i,k\}} r_g\cdot D_g.$$

By Equation 4 and the definition of ϕ_g, the above equation becomes

$$D=\sum_{g\in\{i,k\}}\frac{r_{i,k}\cdot\phi_g}{E_g-r_{i,k}\cdot\phi_g}.$$

For discussion convenience, we name the elements of $\{i,k\}$ in sequential numbers, and replace the path subscripts by these numbers. Let m be the number of elements in $\{i,k\}$. Then we have

$$D=\sum_{l=1}^{m} r_l\cdot D_l=\sum_{l=1}^{m}\frac{r_{i,k}\cdot\phi_l}{E_l-r_{i,k}\cdot\phi_l},\qquad(9)$$

$$\sum_{l=1}^{m}\phi_l=1$$

The best assignment of ϕ_l's to minimize D is to solve the following equations.

$$\frac{\partial D}{\partial\phi_l}=0,$$

for all $l\in\{1,2,\ldots,m-1\}$. Thus

$$\frac{\partial D}{\partial\phi_l}=\frac{r_{i,k}\cdot E_l}{(E_l-r_{i,k}\cdot\phi_l)^2}+\frac{-r_{i,k}\cdot E_m}{\left[E_m-r_{i,k}\cdot\left(1-\sum_{j=1}^{m-1}\phi_j\right)\right]^2}=0,$$

for all $l\in\{1,2,\ldots,m-1\}$. It implies that

$$\frac{E_l}{(E_l-r_{i,k}\cdot\phi_l)^2}=\frac{E_q}{(E_q-r_{i,k}\cdot\phi_q)^2},\qquad(10)$$

for any l, $q\in\{1,2,\ldots,m\}$. If the traffic load is light, then

$$\left(1-\frac{r_{i,k}\cdot\phi_l}{E_l}\right)^{-2}=1+2\cdot\frac{r_{i,k}\cdot\phi_l}{E_l}+3\cdot\left(\frac{r_{i,k}\cdot\phi_l}{E_l}\right)^2-4\cdot\left(\frac{r_{i,k}\cdot\phi_l}{E_l}\right)^3+\ldots$$
$$\simeq 1+2\cdot\frac{r_{i,k}\cdot\phi_l}{E_l}\quad\text{if } E_l>>r_{i,k}\cdot\phi_l.\qquad(11)$$

By combining Equations 10 and 11, we obtain a very simple solution

$$\frac{\phi_l}{E_l}=\frac{\phi_q}{E_q}\qquad(12)$$

for any q and $l\in\{1,2,\ldots,m\}$. According to Equations 9 and 12, we have

$$\phi_l=E_l/\sum_{q=1}^{m}E_q\qquad(13)$$

for any $l\in\{1,2,\ldots,m\}$. Equation 13 indicates the best solution of ϕ_l when traffic load is light. Some ϕ_l's are so small that almost no packet generated at node i is transmitted on the corresponding paths. Without degrading the delay performance, we may disable these unused paths by setting $\phi_l=0$ and reassign ϕ_l's as the presented update algorithm does. In other words, the proposed algorithm can assign almost the best solutions to ϕ_l's when the traffic load is light.

If the traffic load becomes heavier, Equation 11 will no longer be true. A new solution is derived by modifying the solution of Equation 13. Let χ_{lq} be the amount that is moved from ϕ_q to ϕ_l for for better assignment. In other words, the new probabilities ϕ_q' and ϕ_l' are obtained by

$$\phi_q'=\phi_q-\chi_{lq},$$
$$\phi_l'=\phi_l+\chi_{lq}.\qquad(14)$$

Let

$$\sum_{w=1}^{m} E_w \equiv E,$$

and

$$\Delta \equiv \frac{\phi_l' \cdot r_{i,k}}{E_l - \phi_l' \cdot r_{i,k}} + \frac{\phi_q' \cdot r_{i,k}}{E_q - \phi_q' \cdot r_{i,k}}.$$

According to Equations 13 and 14, we have

$$\Delta = \frac{(E_l + \chi_{lq} \cdot E) \cdot r_{i,k}}{E \cdot E_l - (E_l + \chi_{lq} \cdot E) \cdot r_{i,k}} + \frac{(E_q - \chi_{lq} \cdot E) \cdot r_{i,k}}{E \cdot E_q - (E_q - \chi_{lq} \cdot E) \cdot r_{i,k}}.$$

The best χ_{lq} to minimize Δ is obtained by solving the following equation:

$$\frac{\partial \Delta}{\partial \chi_{lq}} = \frac{r_{i,k} \cdot E^2 \cdot E_l}{[E \cdot E_l - (E_l + \chi_{lq} \cdot E) \cdot r_{i,k}]^2} - \frac{r_{i,k} \cdot E^2 \cdot E_q}{[E \cdot E_q - (E_q - \chi_{lq} \cdot E) \cdot r_{i,k}]^2} = 0$$

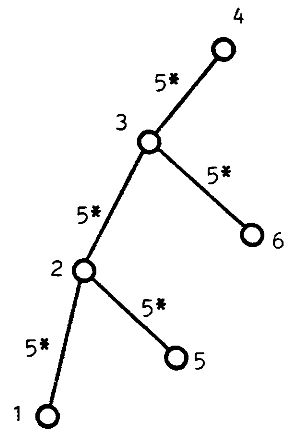
Solving the above derivative equation, we obtain

$$\chi_{lq} = \frac{E - r_{i,k}}{E \cdot r_{i,k}} \cdot (\sqrt{E_l} - \sqrt{E_q}) \Big/ \left(\frac{1}{\sqrt{E_l}} + \frac{1}{\sqrt{E_q}}\right). \quad (15)$$

The solution obtained by Equation 15 implies that we can expect better performance if χ_{lq} is moved from ϕ_q to ϕ_l. In other words, if $\chi_{lq} \simeq 0$, no adjustment between ϕ_q and ϕ_l can improve the delay performance. Suppose that $E_l \simeq E_q$ for any l and q; the set of ϕ_l's obtained by Equation 13 is almost the best assignment. In the PPD method, the paths with small effective capacities are not used. It implies that $\chi_{lq} \simeq 0$, where l and q belong to the available paths. Equation 15 also shows that the greater $r_{i,k}$ is, the less χ_{lq} will be. If $r_{i,k}$ approaches E, the best solution will approach Equation 13. Our update algorithm assigns the best set of ϕ_l's when the traffic is heavy.

6. SIMULATION RESULTS

In the PPD method, the effective path capacity is used to predict the average path delay. It is necessary to verify whether the experimental path delay can be approximated by Equation 4. Figure 6 shows the network used for measuring the average path delay $D_{(1,2,3,4)}$ at a different packet generation rate $r_{1,4}$, ranging from 1 to 10 packets/sec. The packets are generated in a Poisson distribution. Let $r_{5,4} = r_{6,4} = 4$ packets/sec. Assume that every packet has the same packet length. Each packet requires 0.05 sec to pass through any branch. The average path delay $D_{(1,2,3,4)}$ under the different rate $r_{1,4}$ is recorded when the average delay becomes stable. Usually it takes 40 sec to have a stable $D_{(1,2,3,4)}$. The result of the experiment is shown in Figure 7(a). The effective path capacity $E_{(1,2,3,4)}$ for each $r_{1,4}$ is calculated by Equation 5 and shown in Figure 7(b). The capacity for greater $r_{1,4}$ is larger than the physical path capacity. The reason is that the longer-delay case appears with less probability, especially in a heavy traffic situation. The result of the experiment is underestimated because of the short duration of the experiment (time = 40 sec). In Figure 7(a), the average delay is calculated



$\gamma_{5,4}$ = 1/25 Packets/Time-Unit

$\gamma_{6,4}$ = 1/25 Packets/Time-Unit

$\gamma_{1,4}$ is a variable

* The number is the time required to transmit a packet on the branch.

Figure 6—A sample network used for demonstrating path capacity and path delay ($r_{5,4} = r_{6,4} = 4.0$ packets/sec)

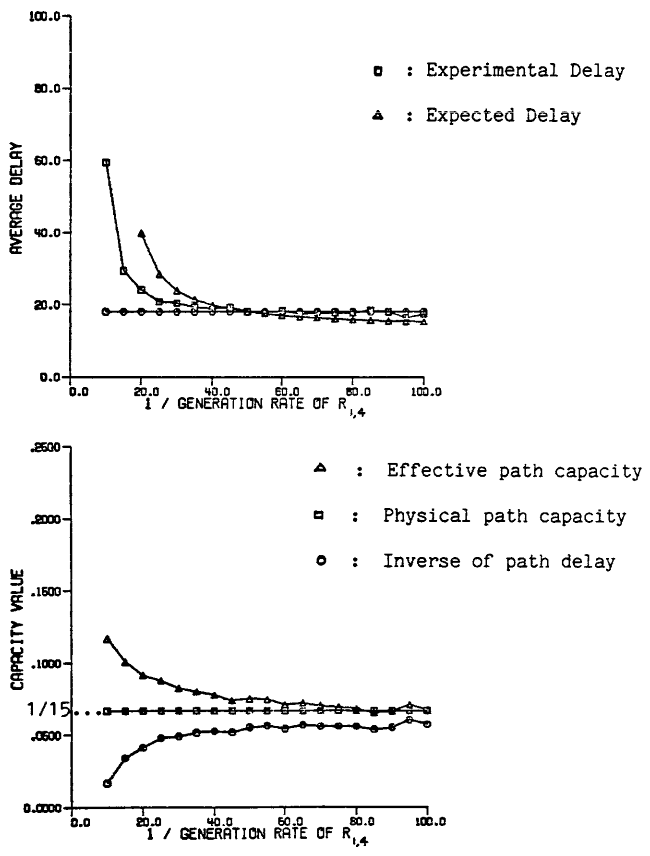
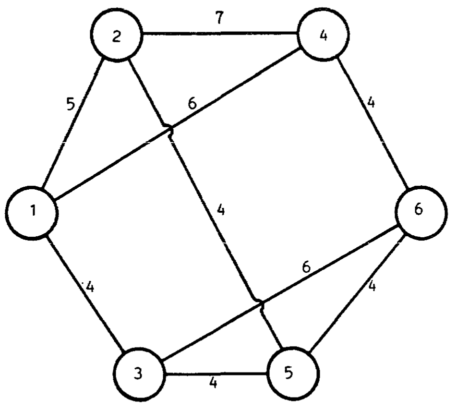


Figure 7—Path delay and effective path capacity



* The numbers on branches are the inverse of corresponding branch capacities in time-units/packet.

Figure 8—A sample computer network for performance simulation experiments

by substituting Equation 4 with the path capacity, evaluated under $r_{1,4} = 2$ packets/sec. The curve is almost the same as the experimental one when the network is not saturated. Although the difference becomes larger in a saturated traffic situation, the tendency of the experimental curve is approximately predicted. Furthermore, the property of this curve is exactly predicted by Equation 8.

We compare the PPD method with the ARPANET method by inspecting their delay performances. The network used for the comparison experiment is shown in Figure 8. Two sets of generation rates are listed in Table III. The entry of each table indicates the variation range of the generation rate. To simplify the comparison process, only the waiting delay and transmitting time are considered, in spite of the shorter overhead of the PPD method. Assume that each packet has the same length. The average delay is calculated over all packets routed in the network. At each node we initialize one packet for one destination to test the adjustment ability of the methods compared.

We first compare the two methods with same update period, i.e., 5 sec in the ARPANET method and $B' = 500$ in the PPD method. The ARPANET method uses global branch delay information to find the shortest distance trees, whereas the PPD method uses the path capacities to determine the assignment probability of each path. We first set $m = \infty$. Figure 9 shows the experimental results under two sets of generation rates. The ARPANET method has a higher average delay than the PPD method. Besides, the ARPANET method

TABLE IIIA—The test set of packet generation rates for light traffic conditions

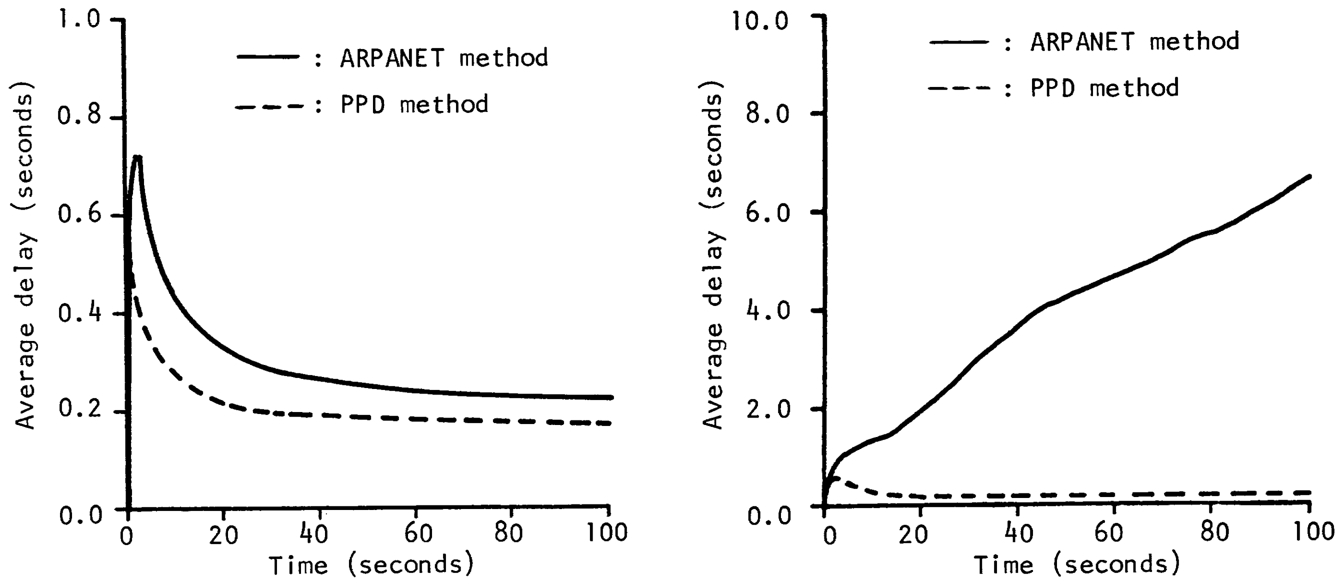
SOURCE \ DESTINATION	1	2	3	4	5	6
1		$\frac{1}{200}\sim\frac{1}{50}$	$\frac{1}{400}\sim\frac{1}{100}$	$\frac{1}{500}\sim\frac{1}{125}$	$\frac{1}{300}\sim\frac{1}{75}$	$\frac{1}{700}\sim\frac{1}{175}$
2	$\frac{1}{100}\sim\frac{1}{25}$		$\frac{1}{500}\sim\frac{1}{125}$	$\frac{1}{600}\sim\frac{1}{150}$	$\frac{1}{400}\sim\frac{1}{100}$	$\frac{1}{1000}\sim\frac{1}{250}$
3	$\frac{1}{600}\sim\frac{1}{150}$	$\frac{1}{300}\sim\frac{1}{175}$		$\frac{1}{400}\sim\frac{1}{100}$	$\frac{1}{500}\sim\frac{1}{125}$	$\frac{1}{200}\sim\frac{1}{50}$
4	$\frac{1}{300}\sim\frac{1}{75}$	$\frac{1}{400}\sim\frac{1}{100}$	$\frac{1}{200}\sim\frac{1}{50}$		$\frac{1}{600}\sim\frac{1}{150}$	$\frac{1}{700}\sim\frac{1}{175}$
5	$\frac{1}{200}\sim\frac{1}{50}$	$\frac{1}{100}\sim\frac{1}{25}$	$\frac{1}{400}\sim\frac{1}{100}$	$\frac{1}{700}\sim\frac{1}{175}$		$\frac{1}{1000}\sim\frac{1}{150}$
6	$\frac{1}{400}\sim\frac{1}{100}$	$\frac{1}{500}\sim\frac{1}{125}$	$\frac{1}{600}\sim\frac{1}{150}$	$\frac{1}{200}\sim\frac{1}{50}$	$\frac{1}{100}\sim\frac{1}{25}$	

* The unit of entries is packet/time-unit.

TABLE IIIB—The test set of packet generation rates for heavy traffic conditions

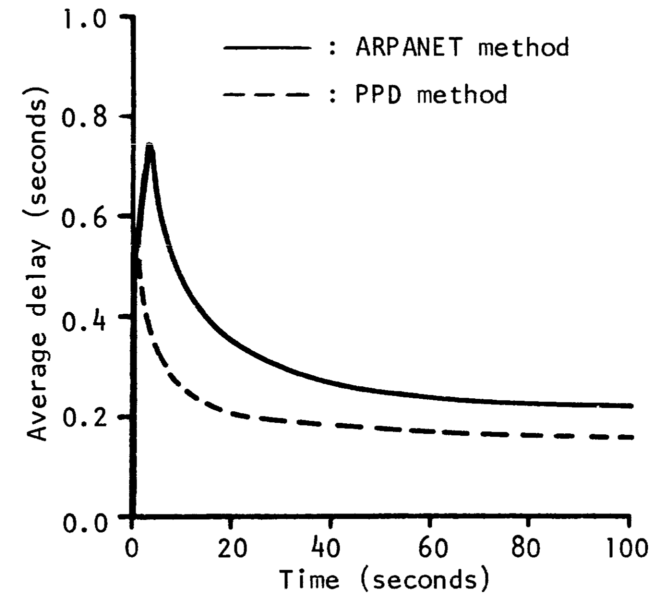
DESTINATION / SOURCE	1	2	3	4	5	6
1		$\frac{1}{100} \sim \frac{1}{25}$	$\frac{1}{200} \sim \frac{1}{50}$	$\frac{1}{248} \sim \frac{1}{62}$	$\frac{1}{148} \sim \frac{1}{37}$	$\frac{1}{348} \cdot \frac{1}{87}$
2	$\frac{1}{48} \sim \frac{1}{12}$		$\frac{1}{248}\ \frac{1}{62}$	$\frac{1}{300}\ \frac{1}{75}$	$\frac{1}{200} \sim \frac{1}{50}$	$\frac{1}{500} \sim \frac{1}{125}$
3	$\frac{1}{300} \sim \frac{1}{75}$	$\frac{1}{148} \sim \frac{1}{37}$		$\frac{1}{200} \sim \frac{1}{50}$	$\frac{1}{248} \sim \frac{1}{62}$	$\frac{1}{100} \sim \frac{1}{25}$
4	$\frac{1}{148} \sim \frac{1}{37}$	$\frac{1}{200} \sim \frac{1}{50}$	$\frac{1}{100} \sim \frac{1}{25}$		$\frac{1}{300} \sim \frac{1}{75}$	$\frac{1}{348} \sim \frac{1}{87}$
5	$\frac{1}{100} \sim \frac{1}{25}$	$\frac{1}{48} \sim \frac{1}{12}$	$\frac{1}{200} \sim \frac{1}{50}$	$\frac{1}{348} \sim \frac{1}{87}$		$\frac{1}{500} \sim \frac{1}{125}$
6	$\frac{1}{200} \sim \frac{1}{50}$	$\frac{1}{248} \sim \frac{1}{62}$	$\frac{1}{300} \sim \frac{1}{75}$	$\frac{1}{100} \sim \frac{1}{25}$	$\frac{1}{48} \sim \frac{1}{12}$	

* The unit of entries is packet/time-unit.

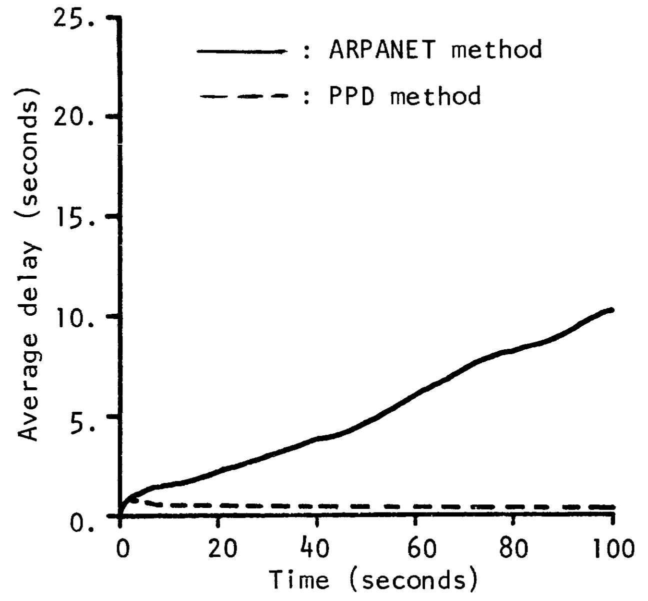


(a) For light traffic. (b) For heavy traffic.

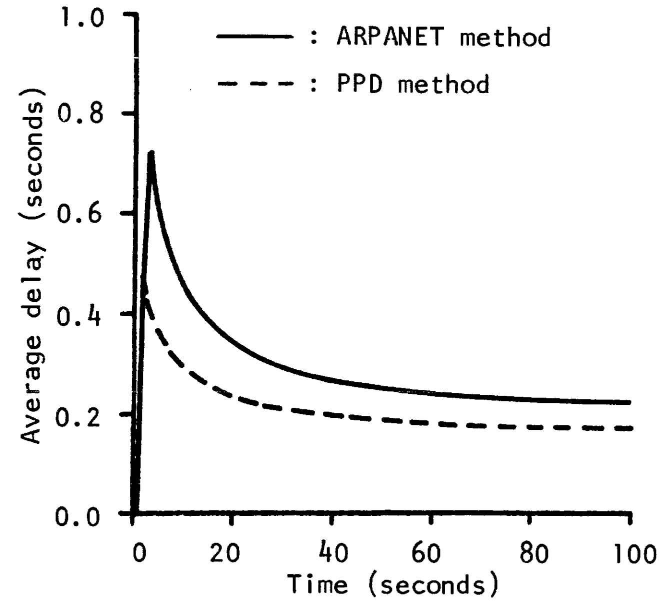
Figure 9—Delay performance of the ARPANET routing and of the PPD routing methods (update period: 5 sec)



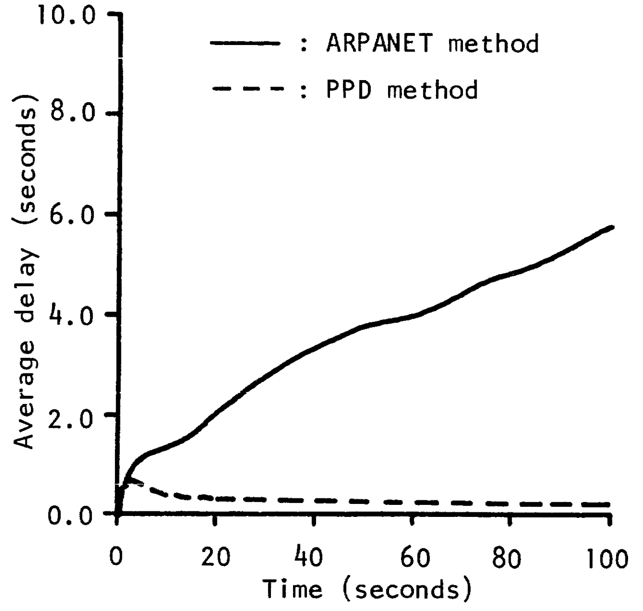
(a) For light traffic (Update period: 2 seconds)



(b) For heavy traffic (Update period: 2 seconds)



(c) For light traffic (Update period: 10 seconds)



(d) For heavy traffic (Update period: 10 seconds)

Figure 10—The performance of the new ARPANET routing and the PPD routing methods

is unable to route the packets under heavy traffic, as shown in Figure 9(b). It shows that the PPD method has more ability to handle the heavy traffic condition. Similar results can be found in Figure 10 when the update periods change to 2 to 10 sec.

We now investigate how a different update period B' affects the delay performance in the PPD method. Three periods—200 time-units, 500 time-units, and 1,000 time-units—are selected for the experiment. Figure 11(a) shows that the three cases perform almost the same except that the shorter B' contributes a faster response to an extreme alteration of the traffic load. Similarly, we examine the delay performance affected by factor B. Two cases with different values of B, 5 and 10, have been simulated. Figure 11(b) shows that longer B contributes better performance. The reason is that the case of longer B collects more delay information to allow better expectation. However, the case of shorter B responds more quickly to the extreme alteration of the traffic load, as shown in Figure 11(b).

The limitation m, which is the maximum number of path entries in a subtable, affects the delay performance much more than the factors B and B'. Figure 11 shows the delay curves with limitations ∞, 6, and 4, under different generation rates (see Tables IIIA and IIIB). It is obvious that smaller m degrades delay performance, especially under a heavy traffic load. Although the parameters A, B, B', and m affect the delay performance, the PPD method is still superior to the new ARPANET routing method, especially in the heavy traffic situation. The reasons are two. First, the PPD method is probabilistic. Second, the PPD method uses the effective path capacities to predict the path delay. The efficacy of the capacities is also confirmed by the experiment.

7. GENERALIZATION OF PPD ROUTING

The update process of the new ARPANET routing method selects the shortest path from node to node. In other words, the new ARPANET routing method operates as a deterministic method between two consecutive updates. Therefore, the ARPANET routing method cannot compete with the proposed probabilistic method, even though the shorter overhead of the PPD method has not been considered in the comparison experiment. To upgrade the delay performance of the new ARPANET method, we generalize it with the PPD approach.

The generalized ARPANET method uses the same delay measurement and update method as its original method.[10] It periodically updates all routing tables at the same time, based on the same delay information. Instead of one shortest path, m shortest paths from one node to any other one are selected during the table update process. The packets, destined for the same node, are probabilistically distributed among the m paths, based on the path assignment probabilities (ϕ's). The entire routing path of a packet is determined by its source node. The generalized ARPANET method uses the algorithm described in Section 3 to route the packets. It assigns ϕ's in proportion to the inverse of the corresponding path delays. In other words, the generalized routing method uses the reverse of the path delays as the effective capacity of the corresponding path.

Only delay time and transmitting time are considered to simulate the generalized ARPANET routing method. The network and generation rates are the same as in Section 6. With $m = \infty$, Figure 12 shows the different delay performance of three routing methods—New ARPANET, PPD, and generalized—under ARPANET routing. It indicates that the ARPANET routing method has the worst delay performance. The PPD methods are slightly better than the generalized method in a heavy traffic condition, as shown in Figure 12(b). It will become more significant if traffic becomes saturated.

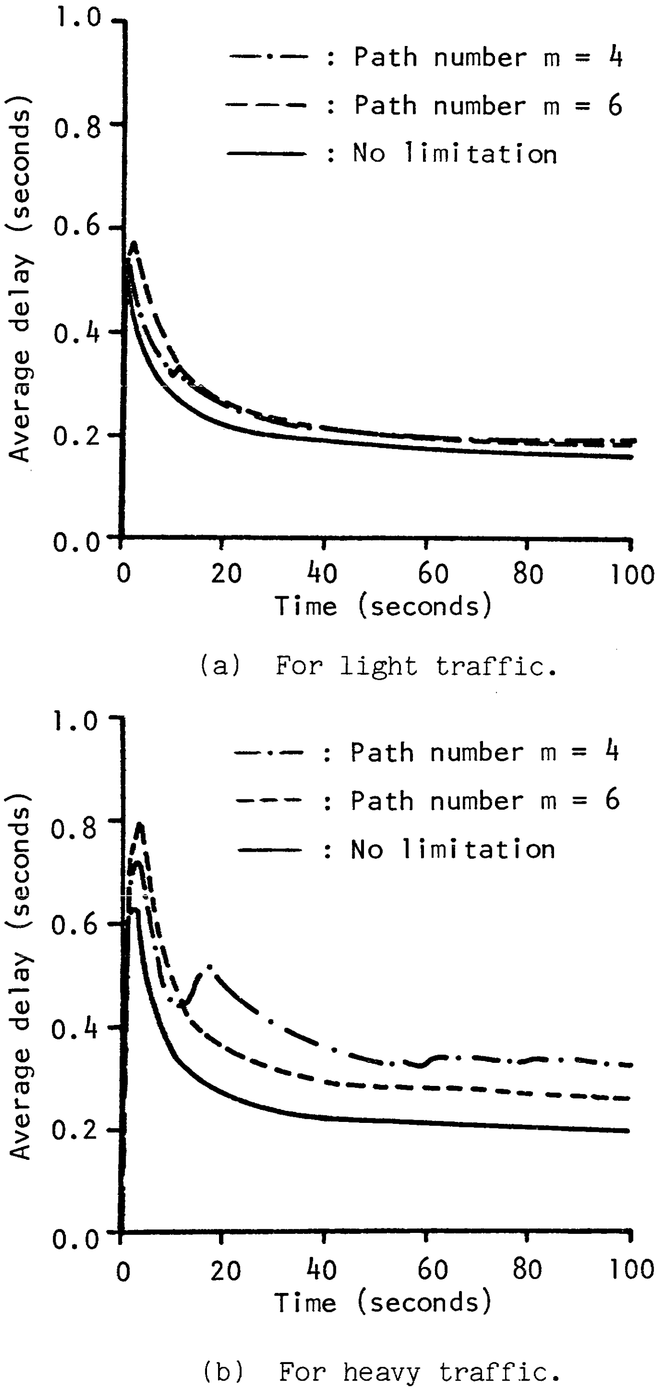
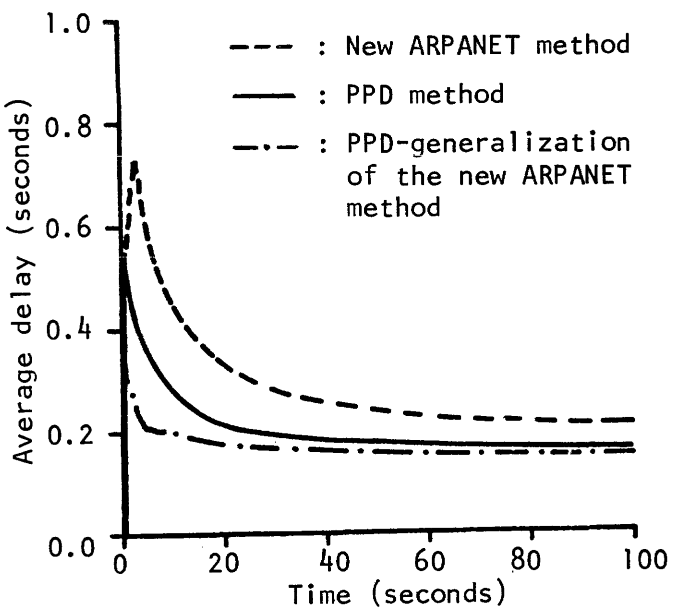


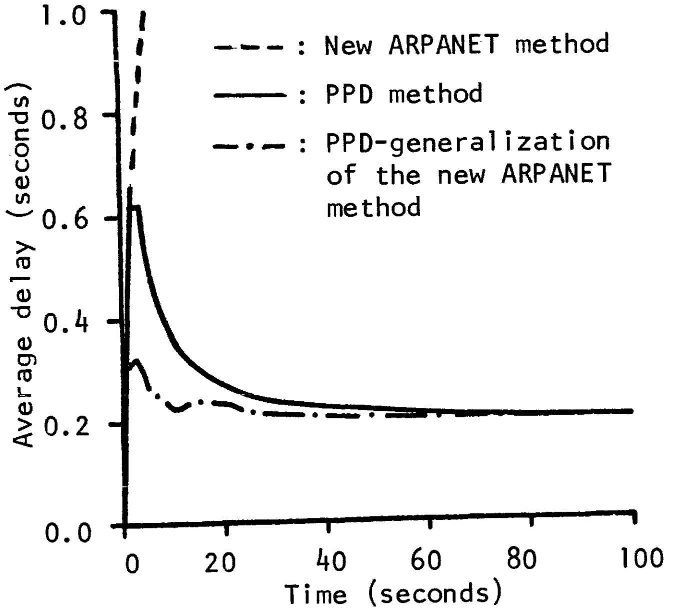
Figure 11—Delay performance of the PPD routing method with bounded path entries in each subtable (update period: 5 sec)

The reason is that the generalized method cannot predict the tendency of delay performance under the saturated condition. However, the method responds well to the extreme alteration of the traffic load, as shown in Figure 12, during the time interval $0 < t < 20$ sec.

In reality, m cannot be infinite. We now investigate how different m affects the delay performance. The delay per-



(a) For light traffic.



(b) For heavy traffic.

Figure 12—Delay performance of the PPD, of the PPD generalization of the new ARPANET, and of the new ARPANET routing methods (update period: 5 sec)

formance with two different m's, 6 and 4, is shown in Figure 13. The PPD method operates better than the other two when the value of m decreases. This conclusion becomes more significant when the traffic load increases. The reason is that the PPD method can collect more delay information for every

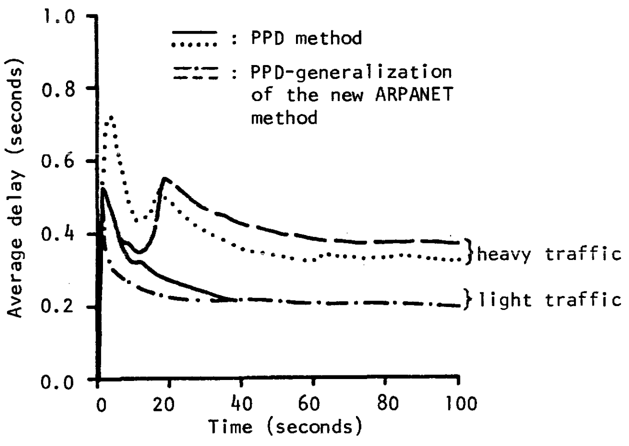


Figure 13—Delay performance comparison of various routing methods with bounded path entries ($m = 4$) in each subtable (update period: 5 sec)

path when the number of paths decreases, although the generalized ARPANET routing methods sometimes perform better than the PPD method.

8. CONCLUSIONS

For the first time, a probabilistic routing method that is path-directed is proposed for computer networks with distributed control. The proposed PPD method can be implemented with small update overhead and minimum node delay. The effectiveness of this method is supported by both analytical and simulation experiments. The method is shown to be more efficient than the new ARPANET routing method, especially in heavy traffic situations. The PPD method is designed to balance the load among multiple paths and to reduce congestion in heavy traffic. The PPD concept can also be applied to a centrally controlled network with some modifications. The results can be extended to developing packet-switched interconnection networks if there are multiple paths between a source and a destination in the hardwired network.

ACKNOWLEDGMENTS

This research was supported by NSF under Grant MCS-80-16580.

REFERENCES

1. Boorstyn, R. R., and A. Livne. "A Technique for Adaptive Routing in Networks." *IEEE Transactions on Communications,* Com-29 (1981), pp. 474–480.
2. Brandt, G. J., and G. J. Chretien. "Methods to Control and Operate a Message-Switching Network." *Proceedings of the Symposium on Computer-Communications Networks and Teletraffic,* Polytechnic Institute of Brooklyn, 1972, p. 263.
3. Cantor, D. G., and M. Gerla. "Optimal Routing in a Packet-Switched Computer Network." *IEEE Transactions on Computers,* C-23 (1974), pp. 1062–1069.
4. Davis, D. W., and D. L. Barber. *Computer Networks and Their Protocols.* New York: John Wiley & Sons, 1979, pp. 89–107.

5. Frank, H., R. E. Kahn, and L. Klenrock. "Computer Communication Network Design—Experience with Theory and Practice." *Proceedings of the Spring Joint Computer Conference,* 1972, p. 225.
6. Fultz, G. L. "Adaptive Routing Techniques for Message Switching Computer Communication Networks." Report, UCLA-ENG-7352, University of California, Los Angeles, July 1972.
7. Gallager, R. G. "A Minimum Delay Routing Algorithm Using Distributed Computation." *IEEE Transactions on Communications,* Com-25 (1977), pp. 73–85.
8. Hwang, K., W. J. Croft, G. H. Goble, B. W. Wah, F. A. Briggs, W. R. Simons, and C. L. Coates. "A Unix-Based Local Computer Network with Load Balancing." *Computer Magazine,* April 1982, pp. 55–66.
9. Kleinrock, L. *Communication Networks: Stochastic Message Flow and Delay*. New York: McGraw-Hill, 1964.
10. McQuillan, J. M., I. Richer, and E. C. Rosen. "The New Routing Algorithm for the ARPANET." *IEEE Transactions on Communications,* Com-28 (1980), pp. 711–719.
11. Naylor, W. E. "A Loop-Free Adaptive Routing Algorithm for Packet Switched Networks." *Proceedings of the 4th ACM/IEEE Data Communications Symposium.* Piscataway, N.J.: IEEE, 1975.
12. Ni, L. M., and K. Hwang. "Optimal Load Balancing Strategies for a Multiple Processor System." *Proceedings of the Tenth International Conference on Parallel Processing,* Belaire, Michigan, August 1981, pp. 352–357.
13. Price, W. L. "Adaptive Routing in Store-and-Forward Networks and the Importance of Load Splitting." *Proceedings of the IPIP Congress 77,* 1977, p. 309.
14. Rosen, E. C. "The Updating Protocol of ARPANET's New Routing Algorithm." *Computer Networks,* 4 (1980), pp. 11–19.
15. Schwartz, M., and C. K. Cheung. "The Gradient Projection Algorithm for Multiple Routing in Message-Switched Networks." *IEEE Transactions on Communications,* Com-24 (1976), pp. 449–456.
16. Schwartz, M., and T. E. Stern. "Routing Techniques Used in Computer Communication Networks." *IEEE Transactions on Communications,* Com-28 (1980), pp. 539–552.
17. Shier, D. R. "Iterative Methods for Determining the k Shortest Paths in a Network." *Networks,* 6 (1976), pp. 205–229.
18. Tanenbaum, A. S. *Computer Networks*. Englewood Cliffs, N.J.: Prentice-Hall, 1981, pp. 196–211.

Optical wireless modem for office communication

by TAKATOSHI MINAMI, KENJIRO YANO, and TAKASHI TOUGE
Fujitsu Laboratories Ltd.
Kawasaki, Japan
and
HISASHI MORIKAWA and OSAMU TAKAHASHI
Fujitsu Limited
Kawasaki, Japan

ABSTRACT

A novel type of optical wireless modem, suitable for office communication, is described.

Two types of modems, *satellite* and *terminal,* each of which consists of head and body, have been developed. The satellite head is usually attached to a ceiling or wall, and the terminal is placed adjacent to a data terminal. The full-duplex mode of data transmission up to 19.2 kbit/s, with the error rate less than 10^{-6}, is realized between the satellite and the terminal which is placed anywhere in the service area of 10 m radius around the satellite head under the fluorescent light.

In this paper, the configurations, specifications, main design features and performances of the modem are described. Some applications are also discussed.

INTRODUCTION

Office automation (OA) systems based on local-area networks or local computer networks have been successfully introduced into various kinds of offices, including the ones in laboratories and factories.[1,2] So the undoubted role of the local networks in OA has been fully confirmed. These networks use either metallic (twisted-pair or coaxial) or optical-fiber cables.

Although local networks based on wired transmission are quite suitable for the trunk line of a network where high speed and reliable transmission is required, wired interconnection is not always the best method for interfacing user terminals and network nodes. The position of office terminals is frequently changed to improve the office layout. Each time, rearrangement of signal cables is also required; this is usually costly and takes a lot of time. In some cases, recabling is so difficult that a new office layout has to be designed under the restriction that changes in the terminal positions are not allowed.

Therefore the use of wireless interconnections for such interfaces is quite suitable, since it avoids the inconvenience of cabling. There are several methods of wireless transmission, among which optical wireless transmission in the infrared region seems to be very attractive for the following reasons.[3]

1. Immunity to electromagnetic interference
2. Little interference with similar systems operating nearby
3. Small size of optical components

In some areas of application other than OA, wireless optical transmission has been widely introduced. Typical examples are a remote controller for TV sets, a cordless telephone,[4] and a wireless speaker system.[5]

The use of optical wireless transmission in offices has also been proposed as a method of interconnecting a cluster of terminals in the same room with a common cluster controller through a diffuse optical channel.[3]

It seems quite likely that increasing needs for a convenient wireless optical interconnection will arise. The specifications for the wireless link in various local networks may be vastly different from one another. Therefore, although it may be desirable to design the wireless optical transmission separately for each network's requirements, it would be more convenient to have versatile types of interconnecting equipment, with standard data terminal interfaces such as the CCITT recommends.

From these considerations we have developed a new type of wireless communication equipment—the "Office-Use Optical Wireless Modem." As a first product we have developed a type of optical wireless modem that is capable of the full-duplex mode of data transmission up to 19.2 kbit/s for the transmission distance of 10 m.

In this paper, the configurations, specifications, main design features, and performance of the modem are described. Some applications are also discussed.

CONFIGURATIONS

Two types of optical wireless modems, *satellite* and *terminal,* have been developed. Both satellite and terminal consist of two parts, *head* and *body*; Figure 1 shows the satellite, Figure 2 shows the terminal. Figure 3 shows an example of installation. Figure 4 shows the circuit-block diagram of the modem. The basic circuit constructions of the satellite and the terminal are the same.

Both heads are equipped with optical transmitter and receiver to carry out the two-way optical transmission between satellite and terminal. Since different subcarrier frequencies are used for up-link (terminal to satellite) and down-link (satellite to terminal), the full-duplex mode of operation is realized. The satellite head can be attached to either the ceiling or the wall of a room, whichever is convenient. Terminals may be put adjacent to office data terminals, such as personal computers or word processors.

Optical waves transmitted from the satellite are non-directionally spread in the room. On the other hand, the terminal head emits an optical beam of medium directivity. Only a rough alignment between the satellite head and the terminal head is necessary. As long as the direct line of sight is not interrupted, terminals can be placed anywhere within 10 m from the satellite head and still achieve a sufficient quality of transmission.

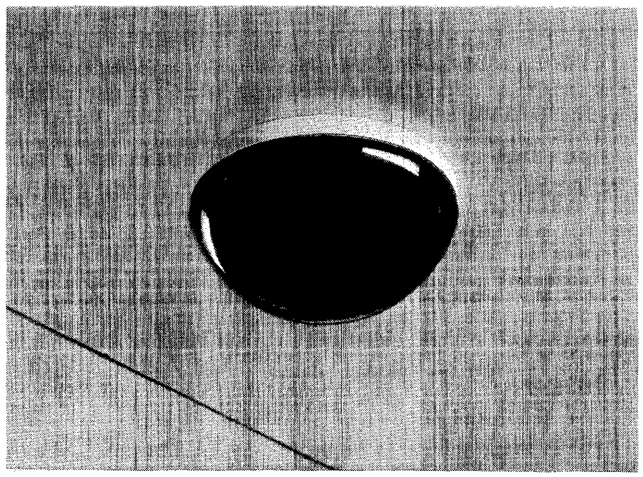


Figure 1—Satellite head

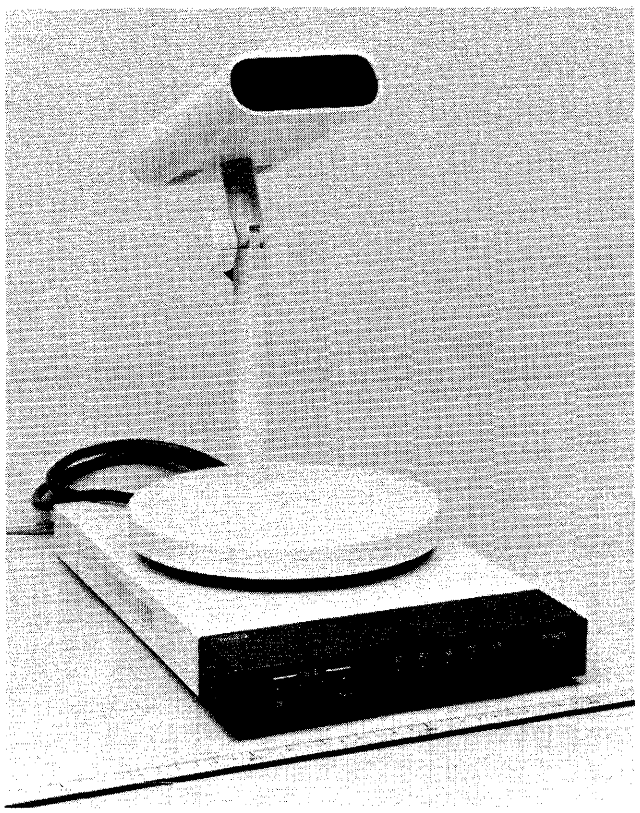


Figure 2—Terminal

SPECIFICATIONS AND DESIGN

The main specifications of the modem are given in Table I. To achieve the specifications, several design considerations were made as follows.

Type of Optical-Wave Propagation

Two types of optical-wave propagation were compared. One is direct line-of-sight transmission and the other is diffuse optical propagation.[3]

Diffuse optical propagation is a type of optical-wave propagation by which the optical waves are spread as uniformly as possible in the room, making use of the reflections of walls and ceilings as shown in Figure 5(a), so that neither a direct line of sight nor alignment between the optical transmitter and receiver is required.

Although the use of the diffuse optical channel is quite convenient, there are several disadvantages to using this method for practical equipment. Since the diffuse optical propagation is based on the reflections at walls and ceilings of a room, each time the optical wireless system using the diffuse propagation is introduced into a new room, a new design is required in terms of the position of transmitters, required optical output power, and directions and directivity of optical emitters that will minimize the optical power required for the transmission to be realized in the whole desired service area. Since the reflection properties of walls and ceilings vary considerably from material to material, required optical power is also quite different from room to room even if the room size is the same.

Furthermore, in some cases the reflections of walls and ceilings cannot be used. One example of such cases is a big hall that is divided into several portions, in each of which different optical wireless systems are to be operated.

In this modem, the direct line of sight transmission shown in Figure 5(b) is employed. Using this type of propagation, the predetermined maximum communication distance is always assured, regardless of the shape or the reflective properties of walls and ceilings, so long as the direct line of sight is not interrupted. The maximum transmission length requires the alignment of the satellite and the terminal. However, since the

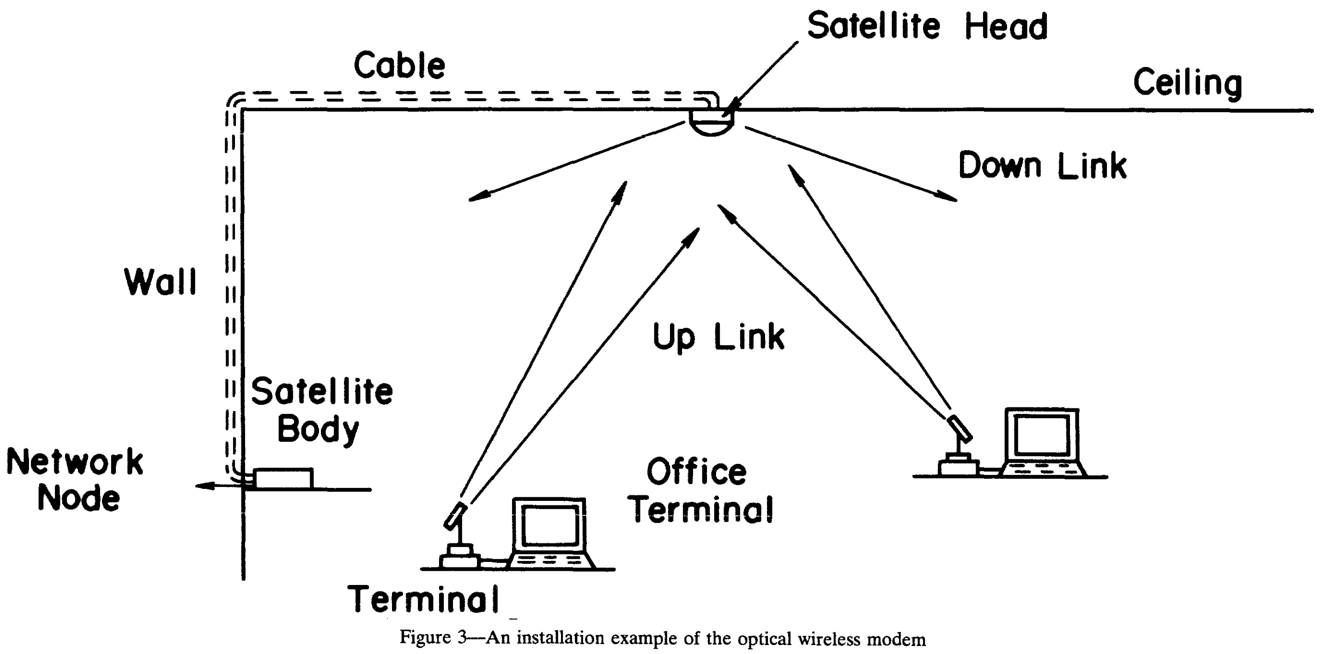


Figure 3—An installation example of the optical wireless modem

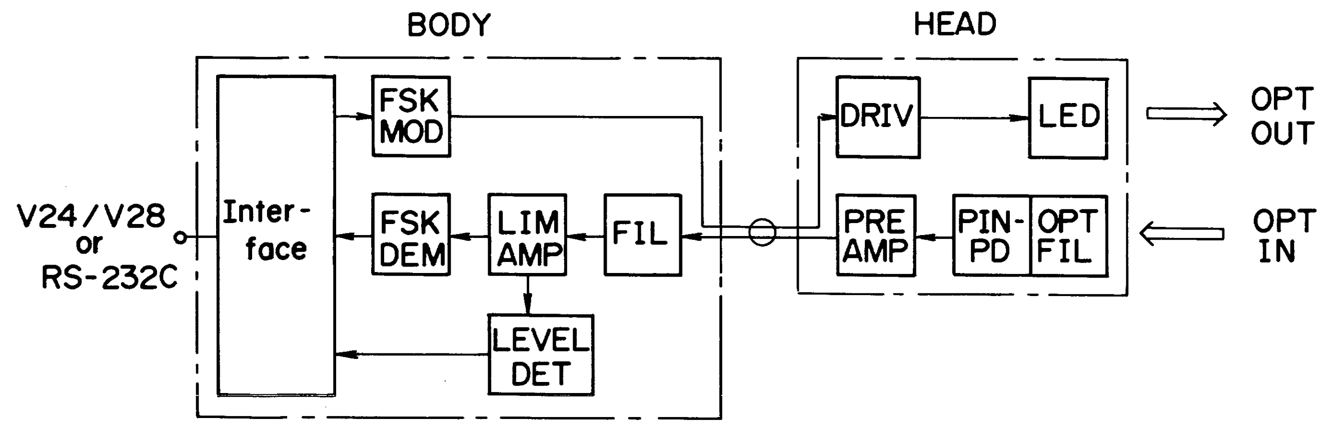


Figure 4—Block diagram of satellite and terminal

optical beam used for the up-link is of medium directivity, no fine alignment is needed.

Furthermore, when the modem is used in a small room that is surrounded by walls, the direct line of sight is not necessarily required, since the optical paths between satellite and terminal are usually assured by the reflections at walls as well as at ceilings, as shown in Figure 5(c).

Ambient Light

The ambient light is a kind of noise source to the optical receiver. The main source of ambient light in office rooms is the lighting, such as fluorescent light.

To avoid the degradation due to the fluorescent light, two kinds of measures are taken, one in the optical domain and the other in the electrical domain.

In the optical domain, by the use of a film type of optical filter attached on the surface of the photodiode, optical power in the region of shorter wavelengths than that of the light-emitting diode (LED) emission is rejected.[3] The optical power in the region of longer wavelength is also rejected, using the property of the silicon PIN photodiode that it has little optical sensitivity in this region.

On the other hand, an experiment was carried out to measure the noise spectrum at the output of the low noise preamplifier with the photodiodes in a room where fluorescent light was used with standard intensity of illumination. The results indicate that noise from the fluorescent lights dominates the receiver noise in the frequency region lower than 300 kHz, even when the optical filter is attached on the photodiodes (see Figure 6). For this reason, the FSK modulation at higher subcarrier frequencies than 300 kHz was adopted. By this measure together with the optical filter, the influence of fluorescent light on the transmission performance is fully avoided.

The effect of sunlight is a somewhat bigger problem, since sunlight has quite a large amount of optical power in the infrared region. When the receiver is exposed to the direct sunlight, the transmission distance is reduced from 10 m to approximately 2 to 3 m. However, it may be rare to receive such strong sunlight in office rooms. Under normal use, the reduction of the transmission distance in a room with windows is usually 10 to 20%. Furthermore, in most offices direct sunlight is usually shaded by a blind or a coating on the window-glass to improve the efficiency of the air conditioning. This also improves the efficiency of the modem use, even near the window.

The direct incidence of strong tungsten light can also cause noise problems, and it should be avoided.

Table I—Main specifications of the optical wireless modem

Item	Specifications
Data rate	up to 19.2 kbit/s
Modulation	FSK-IM
Mode of operation	Full or Half-Duplex
Service area	10 m radius (Error Rate $<10^{-6}$, under Fluorescent Light)
Interface	CCITT V24/V28 or EIA RS-232C
Optical source/detector	LED ($\lambda = 0.88$ μm)/Si PIN-PD
Type of optical wave propagation	
Down-link	Nondirectional radiation
Up-link	Medium-directivity beam
Power supply	AC 100 ±15 V, 50/60 Hz
Power consumption	7 VA
Dimensions (terminal)	210 (width) × 320 (height) × 325 (depth) mm
Environment	Temperature: 5° to 40° C

Full-Duplex Mode of Operation

To realize the full-duplex mode of operation, different subcarrier frequencies are used for the up-link (terminal to satellite) and down-link (satellite to terminal). The subcarrier frequencies are chosen according to the following considerations.

Suppose the subcarrier frequency of the down-link is f_1. The transmitted optical waves from the satellite are reflected from various surfaces, such as floors, walls, and desks. The reflected waves come back into the satellite. So the satellite receives the reflected down-link waves as well as the up-link waves.

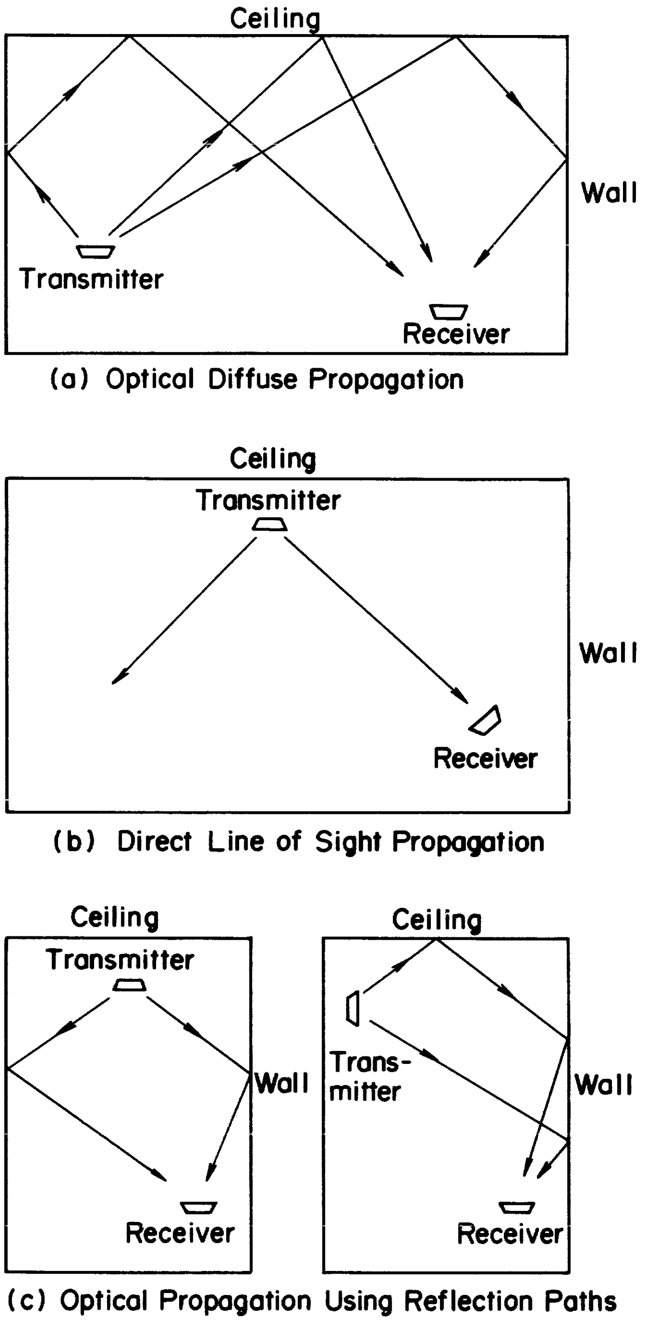


Figure 5—Types of optical wave propagation

The level of the reflected waves varies considerably, depending on such conditions as the height of the ceiling. At the same time, mainly because of the nonlinearity of the LED, the higher-order harmonics of f_1 also enter the satellite receiver. These reflected signals should be filtered out by the electrical filter. The situation is also the same for the terminal head.

Computer simulations and experiments were done to determine what filtering characteristic was required to keep the receiver sensitivity from being degraded by the reflected waves. The results show that more than 60 dB rejection of the unnecessary reflected waves is needed relative to the main

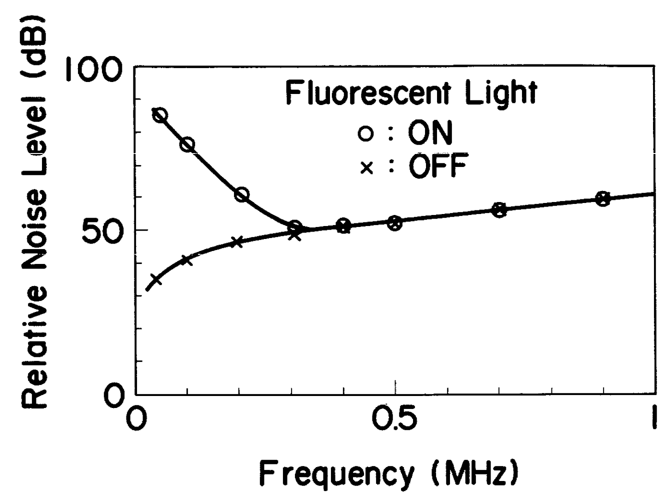


Figure 6—Relative noise level at the output of preamplifier with optical filter

received signal, for the typical case of a satellite attached to a ceiling 3 m high.

At the same time the use of subcarrier frequencies higher than a few MHz should be avoided if the signal is to be free from multipath distortion.[3]

Taking all these factors into account, we chose 1 MHz and 1.5 MHz as the subcarrier frequencies for the up-link and the down-link, respectively. We employ the lower frequency for the up-link in order to minimize the receiver noise, which depends on the product of the subcarrier frequency and the total capacitance of the photodiodes.

Optical Transmitter

High radiant LEDs of 15 milliwatt output power with a directivity of approximately ±60° are used in the optical transmitters in both satellite and terminal heads.

The number of LEDs required depends on the transmission distance. For the satellite head, 9 LEDs were needed to obtain the service area of 10 m radius in rooms with fluorescent lighting. Each of these LEDs is aimed in a different direction so that the optical waves are spread as uniformly as possible.

On the other hand, 5 LEDs were necessary for the terminal head. To save on total optical power, the directivity of the optical beam emitted from the terminal head is adjusted to be about ±30 degrees by a simple rectangular pipe that is put in front of the LEDs. The medium directivity of the up-link beam renders unnecessary the fine alignment that is usually required for optical-space transmission.

Optical Receiver

PIN photodiodes with a typical capacitance of 350 pF and a large active area of 1 cm^2 are used as the optical detector.

In the satellite head, 4 PIN photodiodes are used; they are so aimed as to receive the optical waves coming from all directions in a room. The total capacitance of the photodiodes is approximately 1400 pF. To obtain high receiver sensitivity, the admittance due to this capacitance is canceled by that of the inductance, using a resonance circuit inserted at the front end of the low-noise preamplifier of the receiver; this circuit resonates at the center of the up-link FSK frequencies.

In the terminal head, on the other hand, 2 PIN photodiodes of the same type are used. A similar resonance circuit is also employed.

Other Features

As shown in Figure 1, a hemispheric cover is used on the satellite head to protect the optical components and other electronic circuits from damages, while allowing the optical signal to pass with insertion loss as low as 1.0 dB. The cover also functions as an optical filter to reject visible light. A flat cover made of the same material is used on the terminal head, as shown in Figure 2, for the same reasons.

The satellite body is equipped with a level indicator that tells the input optical power in four steps.

PERFORMANCES AND APPLICATIONS

The modem was actually used in an office environment like that shown in Figure 7.

Figure 7—An office room where optical wireless modem is introduced

It was verified that no direct line of sight was necessary when the transmission distance was less than 3 to 5 m and the wall and ceiling reflections could be effectively used. Desk surfaces and floors could also be used as useful reflectors. However, when the transmission distance was too long, the transmission failed whenever the direct line of sight was interrupted. Such interruptions may be a little annoying. So in situations where a room is shared by a group of people and frequent interruptions of the direct optical path are expected, it is highly recommended that the modem be used with intelligent data terminals, such as personal computers or word processors, that can have an automatic retransmission function with the aid of their software or high-level protocols such as HDLC.

Needless to say, multipoint access is also possible. For example, using a polling access method, where each data terminal is assigned a unique address selected by the network controller connected to the satellite, one satellite can be shared by several terminals around the room.

It sometimes happens that for some reason the satellite head cannot be attached to either ceiling or wall. To cope with such situations, the modem is so designed that terminal-to-terminal communication is also made possible simply by replacing the satellite head by the terminal head and making a minor change in the circuit. Either the direct line of sight or the reflections can be used, as is shown in Figure 8. Since the terminal head has some directivity, terminal-to-terminal transmission may be appropriate for point-to-point communication except in a small room.

CONCLUSION

A novel type of optical wireless modem has been developed and presented in this paper. This modem is quite suitable for office communications, especially for office rooms where frequent layout changes or the portability of the office data terminals is highly required.

Two types of modem, *satellite* and *terminal,* each of which consists of head and body, have been developed. A full-duplex mode of data transmission up to 19.2 kbit/s between

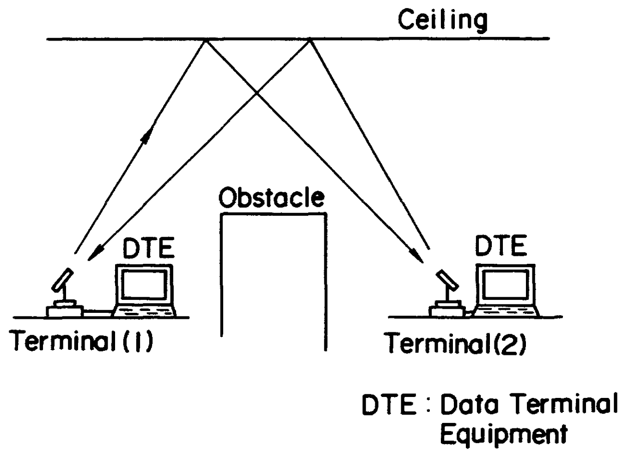


Figure 8—Terminal-to-terminal communication

satellite and terminal is realized in a service area of 10 m radius around the satellite head. This can be attached, for example, to the ceiling of an office, under the fluorescent light. By replacing the satellite head with the terminal head, transmission between two terminal heads is also possible, using either the direct line of sight or the reflections from walls or ceilings.

This modem has been used in an actual office environment, and its feasibility has been confirmed.

ACKNOWLEDGMENTS

The authors wish to thank Dr. H. Takanashi and Messrs. N. Sata and Y. Mochida for their encouragement and valuable advice.

REFERENCES

1. Takahashi, O., Y. Suzuki, and R. Yatsuboshi. "Large Scale Integrated Service Local Network Using Optical Fiber Data Highway." *ICC-81,* pp. 48.2.1–48.2.5, 1981.
2. Matsuda, T., M. Endo, T. Ohyama, and O. Takahashi. "General Purpose Local Network Using Optical Loop Highway." *NTC Record,* pp. G1.2.1–G1.2.5, 1981.
3. Gfeller, F. R., and U. Bapst. "Wireless In-House Data Communication via Diffuse Infrared Radiation." *Proceedings of the IEEE,* 67 (1979), pp. 1474–1486.
4. Braun, E., and S. Schön. "A Cordless Infrared Telephone." Telcom report 3 (1980) No. 2, pp. 83–86.
5. Citta, R. "An Infra-red Wireless Speaker System Utilizing a Super Wide-band FM Carrier." *IEEE Transactions on Consumer Electronics,* CE-21 (1975), pp. 115–119.

A high-throughput interconnection structure

by J. A. HERNANDEZ
Ecole Nationale Supérieure des Télécommunications
Paris, France

E. HORLAIT
Université Pierre et Marie Curie
Paris, France

R. JOLY
Ecole Nationale Supérieure des Télécommunications
Paris, France

and

G. PUJOLLE
Université Pierre et Marie Curie
Paris, France

ABSTRACT

Today, high-throughput interconnection structure (HTIS) is a concept. This paper surveys the properties that HTIS should satisfy and proposes a new architecture that satisfies some of the HTIS requirements. We describe the topology, the access method, and the logical structure of our experimental network. This network is a part of a project called ESCALIBUR, in which both interconnection architectures and distributed applications are studied. The network architecture presented here is mainly oriented toward local and bus networks but may be extended long-haul networks.

INTRODUCTION

The high-throughput interconnection structure (HTIS) can be defined as a digital telecommunications local network capable of supporting a wide spectrum of user needs that require high capacity. Today, such a network exists as a concept; an actual HTIS system has yet to be physically created and its architecture clearly defined.[1,2,3] To satisfy the user's needs, the throughput must be very high. In Figure 1 we give the classical digital throughput for certain services.[4] This figure shows the complex problem of simultaneously satisfying a wide variety of services.

Alarm/security system	4 bits/s
Computer terminal applications	256 bits/s
Digitized telephone speech	32 or 64 Kbits/s
Picture phone	2 Mbits/s
Interprocessor bus	8 Mbits/s
Color television	128 Mbits/s
High-quality video service	512 Mbits/s

Figure 1—Mean bandwidth required for certain services

In this paper we give some new ideas on a possible architecture for HTIS. Then we present a project called ESCALIBUR,[5] an example of a real experience in the area of future networks.

SERVICES AND ARCHITECTURE

Three very different services should coexist in a high-throughput interconnection structure: a telephone network, a video network, and a computer bus. Table I characterizes some of the differences of these networks. The problem is to know what sort of architecture is to be chosen. Some partial responses exist: The baseband local network[6] is capable of supporting a data communications channel[7,8,9] and a very limited number of telephone channels (response time and asynchronous characteristics of these networks are contradictory to a synchronous real-time digital service). The broadband local network provides a communication backbone that can accommodate application. But the words *digital* and *integrated* are not satisfied. CATV technology currently gives the best possibilities. For example, the MITRE coaxial cable LAN has been in operation since 1979. It provides the following services:

1. Twelve video channels
2. One FM radio channel
3. Voice communication
4. One time-division multiple access data bus

TABLE I—Main characteristics of communication networks

Telephone Network	Video Network	Computer Bus
little bandwidth channels	wide bandwidth channels	very high speed transmission
bidirectional	unidirectional	bidirectional
state wide network	building wide network	room wide network
high interactive	noninteractive	highly interactive
real time	real time	non real time

Another response to HTIS is the private automatic branch exchanges (PABX): telephone network and limited computer channels. PABX can support between 64 Kbits/sec and 72 Kbits/sec of channel capacity per user.

In these examples, HTIS concepts are not fully satisfied. However, local HTIS can be looked upon as an extension of a broadband local network and a baseband local network. By the mid-1980s, the HTIS will include circuit-switched, packet-switched, and nonswitched capabilities. These three sets of capabilities are interrelated. In some special cases, they may share common facilities and equipment, but the capabilities are interconnected. Essential characteristics of the architectures of the mid-1980s will include digital transmission through a packet-switched network.

The difficulty of integration comes from the necessity of knowing application characteristics. This may lead to networks having sufficient intelligence to determine whether the customer's information should be packet or circuit switched.

A NEW ARCHITECTURAL CONCEPT

To allow a superposition of telephone, video, and data transfers, a new communication concept would provide the main following characteristics:

1. A medium supporting 1 Gbit/sec.
2. An access to the shared medium allowing real-time communication. For example, a carrier sense multiple-access scheme does not permit realistic telephone traffic.
3. Error detection depending on the customer's information.
4. High-throughput output interfaces.

A project shared by the Institut de Programmation of the Université Pierre et Marie Curie and the Ecole Nationale

Supérieure des Télécommunications in Paris is beginning to analyze and experiment with a new concept of HTIS. This project is called ESCALIBUR.

The communication medium comprises N parallel channels. In a first approach N is equal to 64 parallel lines. The capacity of each line is 1 Mbit/sec. This involves a relatively simple technology. Since the first approach uses telephone cables, the future use of fiberoptics will allow us to choose speedier lines and a larger network.

An important part of this work is to optimize the number of parallel channels and the capacity of each line to reach the 1-Gbit/sec bandwidth.

The new concept provided by ESCALIBUR concerns frame transport. A frame is carried in parallel over the network. The frame structure is classical and comes from the HDLC standard protocol. Two kinds of frames will be used; they are shown in Figure 2.

1	1	1	3	2 (Bytes)
Sender address	Receiver address	Supervisory field	Data	CRC for I frames

Figure 2—Structure of frames

I frames are used for asynchronous information transfer with strong error detection and correction. UI frames can transport asynchronous and synchronous traffic with no error correction (e.g., telephone or video traffic).

Our switching technique is a new one for a local network architecture.[10] It is only an extension of bus principles, however, but is connected with the two first levels of ISO architecture.[11] This switching technique involves a simple access logic to the communication medium. It avoids collisions and allows for a completely decentralized management.

The topology adopted is a ring[12, 13] with a "garbage" unit to avoid problems with erroneous frames running around the network indefinitely. The access method is based on the following two main ideas:

1. We use an extension of the register insertion protocol.[14]
2. A frame is only carried away by its emitter (or by the garbage unit if some error occurred).

The access logic is shown in Figure 3. Access is obtained through five registers to guarantee the absence of collision. Let us describe the three possibilities.

1. Introduction of a frame contained in the sender buffer R5. This is possible only if registers $R2$, $R3$, and $R1$ are free. Nevertheless, the communicator cannot send a frame.
2. Passage of a frame. When register $R1$ receives a frame, a decoder looks at the receiver address field. If the address is not the address of the communicator, the frame is copied on register $R2$ or $R3$, depending on the occupancy of these registers. If the sender address field is

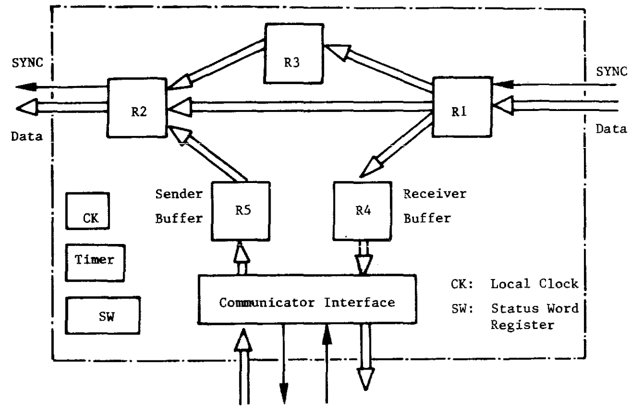


Figure 3—The ESCALIBUR network interface

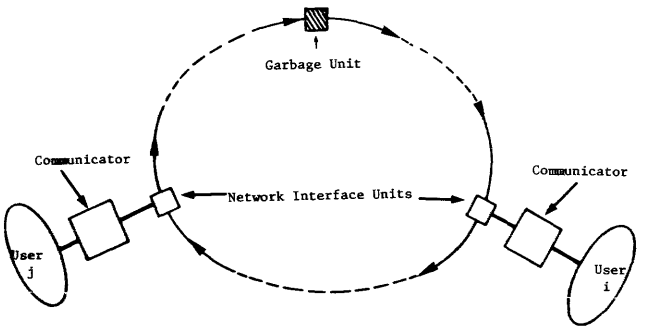


Figure 4—The ESCALIBUR network topology

the address of the communicator, the frame is destroyed in register $R1$.
3. Reception of a frame. If the decoded address is the address of the communicator, the frame is copied on the receiver buffer $R4$ (and also in $R3$ or $R2$ if necessary).

This access logic guarantees a finite transmission delay for a frame ready in $R5$.

The ESCALIBUR project aims to propose a new network able to replace both PABX and the baseband local network. If the second characteristic is satisfied, the first one is more difficult to achieve. In effect, we have chosen an internal packet switching following the X25 packet format to make interconnection with public network easier. Packet switching is not appropriate for digitized telephone speech: The voice is converted into a digital signal by means of pulse-code modulation. This technique requires synchronizations, which are generally incompatible with packet switching. As an illustration we can point to the difficulty of integrating digitized telephone voices on the Ethernet network. For a 10-Mbits/sec local network only 10 or so voice channels can be easily supported. In the ESCALIBUR network a circuit switch possibility is mixed with the packet switch, thanks to the station's capacity to recover empty frames regularly. However, to keep the network asynchronous and to avoid the use of a main station to manage the network, this empty frame can deliver a high-throughput channel (which is larger than a PCM channel but is used either for voice traffic or for data traffic). This possibility, along with a correct buffering system (placed in each node to avoid echo problems), is the reason why synchronous traffic can be transported through ESCALIBUR.

CONCLUSION

For future high-throughput interconnection structure, it is necessary to develop other new concepts and to make major changes in the architecture and capabilities of data communications equipment. A most important element of change will be that users will have a workstation at their desks with a digital phone, a keyboard, and a graphic screen. This necessitates obtaining very high throughput not only for local but also for long-haul networks. Our objective with the ESCALIBUR project is to propose a solution for local HTIS networks. But with the development of fiberoptic cables,[15] our solution can be considered feasible for networks of 50 km.

The problem of a network's determining whether the customer's information should be packet or circuit switched has still to be solved. This facility is contradictory to ISO 7-level architecture: Level 2 should know the parameters of level 7. No satisfactory solution has yet been provided.

REFERENCES

1. Anderson, G. A., and E. D. Jensen. "Computer Interconnection Structures: Taxonomy Characteristics and Examples." *Association for Computing Machinery, Computing Survey,* (1975), pp. 197–213.
2. Penney, B. K., and A. A. Baghdadi. "Survey of Computer Communication Loop Networks: Part 1." *Computer Communications,* 2 (1979), pp. 165–180. Part 2: 2(1979), pp. 224–241.
3. Pierce, J. R. "Network for Block Switches of Data." Bell System Technology Journal, 51 (1972), pp. 1133–1145.
4. Martson W. B., and J. S. Hunter. "CABLE TV: The Missing Link in a National Data Communication Architecture." *Actes des 3ème journées de l'IDATE,* (1981), pp. 75–82.
5. HERNANDEZ, J. A., E. Horlait, R. Joly, and G. Pujolle. "ESCALIBUR—Une Structure d'Interconnexion a Haut Débit." *Proceedings of SEIR 2,* (1982), pp. 425–437, Universidad de Santiago de Compostela, Spain.
6. Clark, D. D. et al. "An Introduction to Local Area Network." *Proceedings of the IEEE,* 66, 11 (1978), pp. 1497–1517.
7. Farmer, W. D., and E. E. Newhall. "An Experimental Distributed Switching System to Handle Bursty Computer Traffic." *Proceeding of the Association for Computing Machinery Symposium on Problems in the Optimization of Data Communication Systems,* Pine Mountain, Ga., (1969), pp. 31–34.
8. Liu, M. T., and G. G. Reames. "The Design of the Distrubted Loop Computer Network." *Proceedings of the 1975 International Computer Symposium* (Vol. 1), 1975, pp. 273–282.
9. Metcalfe, R. M., and D. R. Boggs. "Ethernet: Distributed Packet Switching for Local Computer Networks." *Communications of the Association for Computing Machinery, 19, 7* (1976), pp. 395–404.
10. Horlait, E. "Protocoles de Communication de Bas Niveau et Réseaux Locaux," These de 3ème cycle, Université de Paris sud, Novembre 1981.
11. Zimmermann, H. "OSI Reference Model—The ISO Model of Architecture for Open System Interconnection." *IEEE Transactions on Computer,* (1980), pp. 425–432.
12. Farber, D. J. "A Ring Network." *Datamation,* 21, 2 (1975), pp. 44–46.
13. Saltzer, J. H., and D. D. Clark. "Why a Ring?" *Proceedings of the 7th Data Communications Symposium,* (1981), pp. 211–217.
14. Wilkes, M. V., and D. J. Wheeler. "The Cambridge Digital Communication Ring." *Proceedings of the LACN Symposium,* (1979), pp. 47–60.
15. Pendibidu, J. M. "Présentation du Projet THERESE." *Technique et Science Informatiques,* (1982), pp. 253–257.

A new look at computer contracts

by DENNIS K. KNIGHT, ESQ.
Hitt, Hartwell & Knight
San Diego, California

ABSTRACT

Standard form contracts for computer sales and licenses are typically drafted primarily to protect vendors by limiting their legal liability. These contracts do not usually play a constructive role in the management of a computer project.

Three factors cast doubt on the enforceability of the vendor-protective provisions of these contracts: (a) two recent court decisions refusing to enforce these protective provisions, (b) the application of consumer protection laws to certain computer transactions, and (c) the proliferation of business computer systems among unsophisticated users.

This paper discusses these factors and proposes as an alternative to these standard form contracts a type of contract patterned after system development methodology and designed to educate the customer, provide a management guide for successful computer system acquisition, and reduce the risk of a court's refusing to enforce the contract.

INTRODUCTION

One of the standards adopted by the data processing industry is the general form of contract used for computer sales, leases, and licenses. Typically, these computer contracts are prepared by the vendor's legal staff, printed in somewhat small print on the reverse side of an invoice form, and designed to give the vendor maximum protection and limited risks.

Three factors cast doubt on the enforceability of important parts of these standard contracts:

- Two recent court decisions in which the court refused to enforce certain vendor-protective terms,
- The application of consumer protection laws to certain computer transactions, and
- The proliferation of business computer systems among unsophisticated users.

This paper discusses these factors, their potential effect on contracting practices in the computer industry, and an alternate approach to computer contracts.

CURRENT STANDARD FORM CONTRACTS

Almost all standard form contracts attempt to limit vendor warranties to those explicitly stated within the contract. To the extent that these clauses are enforced by a court, they effectively cancel oral sales representations that are not repeated in the contract. They also, if upheld in court, do away with certain warranties implied by the law. For example, the implied warranty of fitness for a particular purpose provides that if a customer relies on the vendor to provide a system that will do a particular job, if he or she tells the vendor of such reliance, and if the vendor accepts this responsibility by providing a system for that purpose, the law will read into the contract a warranty that the equipment will achieve that purpose.

Standard contracts generally contain language that disclaims implied warranties. The same statutes that set up the implied warranties also provide a means to disclaim them, and the vendor contract forms usually do so in the prescribed manner.

Standard contracts also seek to limit the kind and amount of damages for which the vendor can be held liable. For example, liability for consequential and incidental damages is typically disclaimed. Consequential damages include lost business caused by failure of the computer system. Incidental damages include electrical power and air-conditioning expenses incurred to provide the proper environment for the computer and personnel costs associated with attempted computer usage. The usual goal of such limitations is to avoid potential liability for anything more than the price of the computer system.

Until recently, courts have generally enforced these disclaimers and limitations in cases involving commercial parties, saying that the parties are free to bargain whatever shifting of risk they desire. Therefore, vendors could rest comfortably in the belief that their risk of a judgment greatly in excess of the price of the system was minimal.

RECENT COURT DECISIONS

Two recent decisions should cause vendors to think twice about how well their standard form contracts will actually protect them.

The first case, *Glovatorium, Inc. v. NCR,* has received considerable attention in the computer press. The second case, *A & M Produce Co. v. FMC Corp.*, has not received this media attention. As a matter of fact, *A & M Produce Co. v. FMC Corp.* is not even a computer case. However, the principles of the decision appear directly applicable to computer cases and, taken together with the *NCR* case, cast doubt on the effectiveness of vendor protections found in standard form contracts.

Glovatorium, Inc. v. NCR

The *NCR* case was tried in the Federal District Court for the Northern District of California. The jury found for Glovatorium in the amount of $2.3 million, which included a substantial punitive damage award and NCR appealed to the Ninth Circuit Court of Appeals. The Court of Appeals upheld the award of the jury.

The computer system purchased by Glovatorium, a dry-cleaning company, was an NCR 8200 minicomputer with a software package called SPIRIT. The price of this system was about $50,000; therefore, the award of $2.3 million in total damages was far in excess of the system price and demonstrates the potential risk for a vendor in certain aggravated fact situations. (Large punitive damage awards are not generally given by trial courts and upheld by appellate courts unless the facts showing misrepresentation and other types of fraud are clear and aggravated.)

The standard form contract used by NCR in its sale to Glovatorium contained the standard warranty disclaimers and limitations of liability; however this contract language did not protect the vendor once the facts of fraudulent conduct were proven. The court in effect threw out the protective language and treated the case as though the protective language were not part of the contract.

A & M Produce v. FMC Corp.

The *A & M Produce v. FMC Corp.* case is, in my opinion, potentially more significant than the *NCR* case because facts of fraudulent conduct did not play a large role in the case. Instead, the California state court directly considered the validity of warranty disclaimers and limitations of liability for consequential damages. It found this protective contract language of no force or effect, primarily because the judges simply did not think the contract language was fair in light of sales representations made to A & M Produce and the other circumstances of the sale.

FMC sold a tomato weight-sizing machine to A & M Produce for approximately $32,000. FMC sales representatives convinced A & M Produce that the machine was the right type of equipment for them. The sales contract was a standard vendor-prepared document with typical fine-print provisions on the reverse side of an invoice. This contract excluded all warranties except a warranty that the machine was free of defects, and it excluded any liability for consequential damages. In other words, the contract was, in these respects, quite similar to vendor standard form contracts for computers.

When the machine failed to operate satisfactorily, A & M Produce suffered the loss of a large part of the tomato crop. The jury in the Superior Court trial awarded A & M Produce $255,000 plus $45,000 in attorneys' fees. The Court of Appeal, Fourth District, upheld this verdict.

The courts refused to enforce the warranty disclaimers and limitations of liability, finding that these contract terms involved oppression and surprise and that they reallocated the risks of the bargain in an unreasonable, one-sided manner. Some of the courts' language is instructive as to how they viewed the standard form contract:

> The social benefits associated with freedom of contract are severely skewed where it appears that had the party actually been aware of the term to which he agreed, or had he any real choice in the matter, he would never have assented to inclusion of the term.
>
> One suspects that the length, complexity and obtuseness of most form contracts may be due at least in part to the seller's preference that the buyer will be dissuaded from reading that to which he is supposedly agreeing. This process almost inevitably results in a one-sided contract.
>
> Especially where an inexperienced buyer is concerned, the seller's performance representations are absolutely necessary to allow the buyer to make an intelligent choice among the competitive options available. A seller's attempt, through the use of a warranty disclaimer, to prevent the buyer from reasonably relying on such representations calls into question the commercial reasonableness of the agreement.

Thus, the court effectively erased the protective terms in the contract. Furthermore, after such terms were erased, the court could substitute its own judgment about risk allocation, performance guarantees, and other important contract matters.

The legal principles set out in the *A & M Produce* case are directly applicable to computer cases. In fact, the disparity of technical knowledge between vendor and many customers, and the types of aggressive marketing approaches used today in the computer industry, might well cause a court to be even more protective of a computer buyer than of an agricultural machine buyer.

The purpose of this somewhat detailed summary of the *NCR* and *A & M Produce* cases is not to demonstrate the kinds of situations in which a court might find a contract unfair and unenforceable. The legal principles set out in past cases—precedents—must be applied afresh to each new fact situation. A few changes in the factual circumstances of a case can make a marked difference in the result. Rather, the purpose of this discussion is to suggest that the standard form vendor contracts used in the computer industry may not provide the vendor the protection sought. If a court throws out all of the carefully drafted vendor-protective language found in most computer contracts and substitutes its own judgment as to what is fair and reasonable, the form contract provides nothing but a false and misplaced sense of security.

APPLICATION OF CONSUMER PROTECTION LAWS TO CERTAIN COMPUTER TRANSACTIONS

During the last 10 years, a number of consumer protection laws have been passed by the federal legislature and various state legislatures. These laws are designed to replace the old "caveat emptor" (let the buyer beware) axiom with the notion that government and laws will aid consumers in getting a fair deal.

The Magnuson Moss Consumer Warranty Act is an example of federal consumer law. This act requires certain disclosures to be made to consumers about warranties. It regulates the content of warranties and sets out certain warranty service requirements. It also provides that the Federal Trade Commission (as well as private individuals) can take action to enforce this law.

In California, the Song Beverly Consumer Warranty Act regulates warranties for the benefit and protection of buyers. There are also many other consumer protection laws presently on the books in California.

Many other states have enacted consumer protection laws. All of these laws tend to define *consumer* and *consumer goods* broadly. For example, under federal law, a commercial purchaser of an IBM Selectric typewriter may be considered a consumer purchasing consumer goods. Consequently, a seller may find that consumer protection laws apply to what that seller thought was a commercial transaction.

With the availability of small and relatively inexpensive "personal" computer systems to small business owners and professionals, it is likely that many computer system sales will be classified as consumer sales under federal and state consumer protection laws. One result of such a classification is that standard form computer contracts with their disclaimers of warranty and limitations of liability may be found in violation of these laws and, therefore, unenforceable. Furthermore, computer system vendors may find themselves subject to investigative and enforcement actions by administrative agencies such as the Federal Trade Commission. Anyone who has been on the receiving end of these actions knows that they are highly disruptive and expensive to defend against.

It might be said that the *NCR* and *A & M Produce* cases reflect the application of consumer protection principles to commercial transactions. The courts in those cases certainly refused to impose what they considered an unfair deal on Glovatorium, Inc., and A & M Produce, the commercial consumers in the cases.

THE PROLIFERATION OF BUSINESS COMPUTER SYSTEMS AMONG UNSOPHISTICATED USERS

The substantial reduction in the cost of computer hardware, the availability of mass-marketed off-the-shelf business systems and application software, and extremely aggressive marketing efforts for small computer systems are resulting in the acquisition of computer systems by a large number of small business and professional users. One characteristic of this marketplace is the lack of user data processing sophistication. Many, if not most, of these customers are first-time computer users, and this characteristic creates its own set of problems.

Implementation of a computer system, even a small system, takes a great deal of preparation, training, and user sophistication. Computer system users must understand and accept their responsibilities in acquiring and implementing a system. For example, users must define needs with a high degree of specificity to ensure that the computer vendor knows what kind of functions the system must perform. Users must establish and conduct an effective implementation program to ensure that business functions can carry on during the shift from manual to computer operation. Last, users must be an integral part of the trouble-shooting team when problems occur. Simply calling up the vendor and reporting that the system does not work will not do.

Unfortunately, first-time, unsophisticated computer users are frequently unaware of these responsibilities and do not understand the significant impact that the computer system will have on their business operations. They do not understand that this impact will probably be relatively far greater than the cost of the system in dollars or that they will have a key participatory role in the success or failure of the system.

One of the tasks facing a vendor in this small-computer marketplace is educating the customers in their role and responsibilities during a computer system acquisition; however, the contracting practices of most vendors fail to consider this task. Contracts emphasize protection of the vendor, not education of the customer and management of the project. Now, as discussed earlier, even this protective function is of doubtful effectiveness.

AN ALTERNATIVE APPROACH TO COMPUTER CONTRACTS

Why not look at a computer system contract from a new perspective? Instead of perceiving it as a legal document that no one but the lawyers really understands and that few people take the trouble to read, why not look at the contract as a management tool—a project management plan designed to educate and guide the vendor and customer toward the goal of successful system implementation?

Legal documents are frequently used to educate and guide people in their affairs. For example, corporate by-laws can describe to the shareholders, officers, and directors their rights, duties, and responsibilities. These documents are relatively short and concise directives that, if followed, ensure conformance with laws and duties. They educate as well as set out legal relationships. Well-drawn estate-planning documents, such as trust documents, describe trustees' responsibilities and beneficiaries' rights and responsibilities. By reading these documents, trustees can see what they must do to sell trust property and buy other property. Likewise, a computer contract can be drafted to guide vendor representatives and customers to successful system installation. Such a document could save all parties time and money by providing a management guide that educates and directs effort.

Computer System Development Methodology

Consider the typical computer system development methodology described in various texts on data processing project management. The phases of such a methodology might be as follows:

1. System definition phase.
2. System design phase.
3. System test and acceptance phase.
4. Conversion, training, and implementation phase.
5. Maintenance and support phase.

Use of such a methodology is not uncommon in large systems, but it is uncommon in small systems where, it is reasoned, the system cost does not warrant a conventional development effort.

Small systems may be relatively inexpensive, and price pressures may limit the use of systems analysts and other data processing professionals. The problem is, however, that the impact of a small-business computer system on business operations may be enormous. Consider, for example, the effect failure of an inventory-control system might have on an inventory-intensive business like an auto parts store. Small businesses can fail and have failed because of computer system failure.

It may not be realistic to suggest that a business owner purchasing a $10,000 computer system hire an analyst at $40 to $60 per hour to manage the purchase, although such a decision should be guided more by risks and consequences of failure than by the price of the system. However, at least the business owner should be informed and guided in some way through the complex process of a system implementation. It is certainly in the interests of the computer system vendor to minimize the amount of necessary customer hand holding and the number of trouble calls from the customer—or the risk of a lawsuit.

Implementing Methodology via the Contract

The contract should describe and provide a management tool for implementing the methodology. It should be a project planning document as well as a legal document. No magic words are needed, and technical terms of art should be as prevalent as legal terms of art.

The following discussion is not intended to prescribe the form of the contract, nor does it address all of the legal considerations the contract should cover. The purpose of the ideas set forth is to demonstrate how the contract can contribute to the goal of achieving a successful installation.

System definition

The contract should incorporate specifications of system needs. This can be done by including these specifications in the contract or by referring to other documents, such as a functional specification, and incorporating these documents into the contract. Such a contract provision should, first of all, point up the requirement for a written specification. Often, prospective buyers never objectively and specifically determine their needs, much less write them down. Even in a small system acquisition, a buyer can describe, in writing, the way his or her business functions and its special problems and considerations.

If buyer needs are not specified, there will be a real risk that the vendor does not understand what must be done. Furthermore, there will be no objective way to determine whether or not the system does what it is supposed to do, and if a lawsuit occurs, it will be more difficult and expensive to prove whether or not the system was defective.

Warranties are also much more meaningful if they are set out in this manner. The effect of a functional specification is, in many cases, to warrant that the system will perform the specified functions. Such warranties help buyers by informing them of what the vendor is promising to do. They help the vendor by limiting performance promises to what is contained in the specification, assuming that oral sales representations and written performance specifications are substantially similar.

Furthermore, a court is less likely to throw out or ignore contract terms if they seem fair. Specifying in writing what a system will do seems much fairer than making oral sales promises and then repudiating them in the "fine print" of a contract.

System design

If the system software is to be developed especially for a particular buyer, design specifications are required to inform all parties involved what is to be done and how. Even in the case of off-the-shelf software, the functional specification will provide a way to match buyer needs with a package that will meet those needs. If no package will fit the needs in every respect, necessary changes (either to the software or to the business operations) can be identified more easily if the business operations and needs have been previously described, documented, and incorporated into the contract.

System test and acceptance

This may be one of the most critical phases of a computer system acquisition from both a legal and a technical point of view. If a properly designed, planned, and executed test and acceptance procedure is accomplished, all parties can gain a high degree of assurance that the computer system will do the job it was designed to do. If this phase is done haphazardly, system problems may not surface until the buyer is actually using the system. Problems that arise during the operational use of a computer system are the ones that hurt the most and lead to lawsuits.

My experience in the litigation of computer cases has been that the last step in the acquisition process where problems can be discovered and corrected without serious adverse effects on business operations is this system test phase. Despite the importance of this phase, it is often given little attention and concern. Standard form contracts currently in use are almost invariably designed to get the user to sign off as soon as possible rather than to ensure that a proper test and acceptance procedure is employed.

Consider what a good system test requires. First of all, good functional specifications are needed. Not only must these specifications describe the needs to be met, but also objective measures of satisfaction of those needs must be developed against which to test system performance, whether a separate test specification or simply a comprehensive functional specification. The contract can and should set out the test procedures, the method of objective measurement of performance, the location of the testing, the personnel to be used, the type of data to be used, the procedure for isolating and reporting problems, and other considerations relevant to performance testing. This can be done in the contract itself or in another document that is referred to and incorporated into the contract by such reference.

The contract should also specify the procedure for acceptance of the system by the user. There frequently exists a serious discrepancy between what is called acceptance in the contract (if it is mentioned at all) and the practical requirements of an intelligent acceptance procedure. Perhaps an automobile can be intelligently accepted after a short period of driving it, but computer systems, depending on their size and the nature of the operations they support, may take several weeks to several months before users can be said to have had a reasonable opportunity to determine if their system does what is supposed to do. The law will give them this opportunity.

Rarely, particularly in small systems, does the contract or any other document make adequate provision for a reasonable test and acceptance procedure. The result of this situation is a dispute waiting to happen. Without an adequate test and acceptance procedure, users may not find problems until they have used the system operationally for weeks or months. The disputes that arise in this situation are the kind that frequently lead to lawsuits.

If a dispute does arise, despite the use of a good test and acceptance procedure, it can be resolved with much greater ease if there are objective ways of determining system performance.

Conversion, training, and implementation

The inexperienced buyer of a computer system is frequently not aware of the potential magnitude of the tasks of data conversion, training, and implementation. Even in small systems it may take days or weeks to get through this phase successfully. The contract should address the requirements of data conversion, training of user personnel, and implementation of the system into the user's operations. If these matters are addressed in the contract, and if the contract is used as a management tool to be read and understood, users can be educated and informed of the requirements of this important phase and vendors can gain some degree of assurance that, when they believe the project to be nearly finished, major problems in getting the system running will not arise.

If these matters are not discussed and resolved early, they may create another area of potential dispute. As noted earlier, these disputes may grow into expensive and disruptive lawsuits.

Maintenance and support

This area is sometimes covered by a maintenance contract, usually drafted by the vendor's attorney to protect the vendor. Maintenance contracts that a court sees as unfair or unreasonable are just as likely to be struck down as sales contracts. Furthermore, lack of user knowledge and forethought in this area can be just as destructive to a potentially successful transaction as they can in any other area.

The contract should discuss maintenance and support requirements up front, before these requirements turn into emergencies. For example, a user requirement for one-hour maintenance response can be discussed reasonably when the contract is being negotiated. Such a discussion held 15 minutes after a retail point of sale and inventory system goes down in the middle of the Christmas rush is likely to lead quickly to heated words and disputes with lawsuit potential.

CONCLUSION

This paper has discussed certain court decisions, laws, and marketplace characteristics that affect computer contracts and has proposed an alternative to the standard contract currently in use. This alternative is designed to educate, provide a management tool, prevent problems, and maximize the chance of a successful computer system project.

Viewing the contract as a management device rather than as a legal document with no real bearing on practical considerations will benefit both computer vendors and their customers. Vendors will benefit from certainty in their obligations, education for their customers, and happy and satisfied customers. Buyers will benefit from effective methods to ensure that their needs are defined and met with minimum adverse impact on business operations.

In this age of proliferation of highly sophisticated computer systems into business, professional, industrial, and educational entities, it is high time the industry changed its approach to contracting from a protectionist one to a success-oriented one.

An information system for developing information systems

by BRUCE I. BLUM
The Johns Hopkins University
Baltimore, Maryland

ABSTRACT

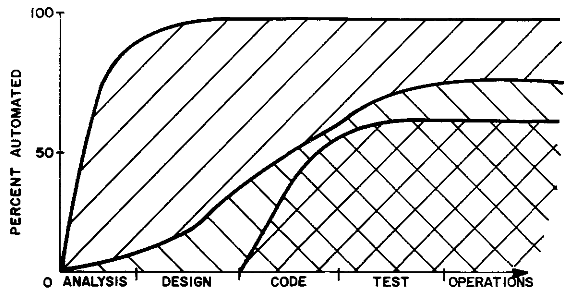
This paper considers a paradigm for the development of software through the use of a complete, integrated-design database. A prototype information system for the implementation of information management systems is then described. This tool manages the requirement definitions, the process and data flow, and all design and operational documentation. Using a relational-data model and program specifications, it generates programs in a selected target language. The results of two years' production use are presented, and the potential for transfer of the concept is examined.

INTRODUCTION

One convenient mechanism for generalizing about progress in computer technology is to speak of generations. Thus, the Japanese are developing a fifth-generation computer[1] that will be an order of magnitude of improvement over the von Neumann computer, and language developers are working on fourth-generation languages[2] that will result in an order of magnitude of improvement in productivity over conventional high-order languages.

If we follow this analogy, we should be looking to a fourth-generation development environment. The first generation required hands-on manipulation of code and data, the second generation used punched cards and their electronic equivalent, and the third generation added libraries of tools and databases to support aspects of the life cycle. In this context, a fourth-generation development environment would provide uniform, integrated support to the full life cycle development process. This includes, of course, the maintenance and configuration management activities that constitute roughly half the total system cost.

Since the end result of computer system development is a machine-sensible product, one would expect practitioners in the field to have developed full-support automated tools. Unfortunately, this is not the case. Source programs are still retained as modified card images in files. Access to and control of programs are normally left to general file management programs. The contents of the program text are usually managed by the compiler with little analyzed data preserved in machine-sensible form. The link loader manages object programs without providing much system structure information. Standalone and integrated packages may manage the database or data dictionaries. Word processing is used to record requirements and specification documentation. Independent tools may support project management, test reporting, configuration management, and so on.



▩ STANDARD DEVELOPMENT CYCLE
▧ NONINTEGRATED ADJUNCT
▨ FULLY AUTOMATED PARADIGM

Figure 1—The role of automation in the production life cycle

Figure 1 illustrates the role of automation in the current production model. Most machine-sensible products are not created until the process of coding begins. Coding is complete (for a given build) before testing begins; the coded product is used during operations. Superimposed upon the coded product is a nonintegrated, machine-sensible adjunct to the product. This adjunct includes text-processed documentation, independent reporting tools, standalone specification data, and so on. Clearly, our goals should be directed toward an integrated support environment that manages all information for each step of the process and links data among development processes. This is called the fully automated paradigm; as shown in Figure 1, it treats the system design information (requirements, specification, code, test data, etc.) as a single integrated database.

In what follows, a prototype fourth-generation development environment is presented. This system, The Environment for Developing Information and Utility Machines (TEDIUM™, a trademark of Tedious Enterprises, Inc.) has been in production use at the Johns Hopkins Medical Institutions for over two years. The tool is directed toward a specific application set: the information management system (IMS).

The IMS is characterized by its use of commercially available hardware and software, its reliance upon a database, and its lack of realtime demands beyond those normally associated with interactive processing. The useful life of an IMS normally is 5 to 10 years; perhaps 75 percent of all commercial EDP activity is related to IMS projects.

The IMS provides a convenient target for a fourth-generation design environment. There is little technical risk associated with a project. That is, because of its reliance upon off-the-shelf products, there is little doubt whether the technology will be able to support the application. There is, however, great application risk—that is, risk that the finished project will not meet the users' needs. This failure to produce acceptable results is a major problem in IMS development. For example, surveys show that 52% of the requested applications are backlogged for two or more years.[3] Further, one report indicated that of the software undertaken, 75% was never completed or not used if completed.[4]

Figure 2 presents an overview of the state-of-the-art with respect to these two axes of risk. While a fully automated paradigm may be appropriate for all system development, it is reasonable to narrow the scope for prototype development. In the case of TEDIUM, the application area is the IMS in general and clinical information systems in particular. The latter manage clinical data in support of patient care. The

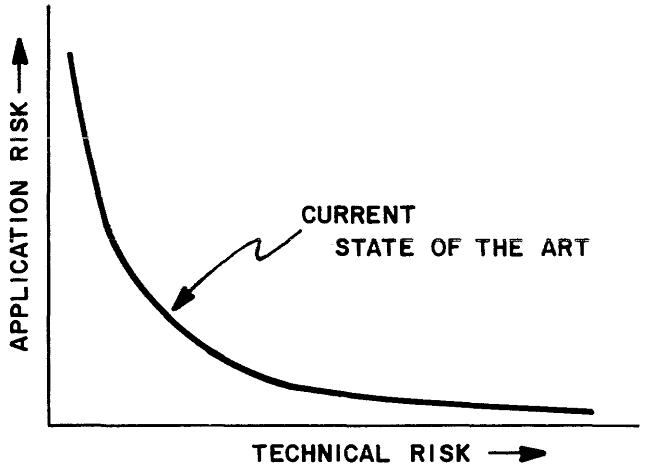


Figure 2—Two axes of development risk

primary challenge is the definition and refinement of requirements in a dynamic user environment. Once the needs are established, there is little difficulty in transforming them into a final product. (Of course, the previously cited performance statistics suggest that although implementation may not be difficult, mere implementation does not imply that there will be a satisfactory product.)

METHOD

TEDIUM is a comprehensive methodology and environment intended to support the development and maintenance of an IMS. It operates upon a database containing all the machine-sensible information about the target IMS. In this sense, it is an information system for developing information systems. As an IMS, TEDIUM is implemented using TEDIUM.

The target applications for which TEDIUM was designed are moderate to large IMSs with heavy user interaction, a limited understanding of the final requirements, and sophisticated data structures. These are custom-built systems that require from one-half to twenty man-years of effort. Though TEDIUM clearly has the ability to act as a report generator or "programmerless" application generator, its design capabilities are considerably broader. Various aspects of the system and its implementation techniques have been described in detail elsewhere. The remainder of this section contains an overview of TEDIUM and provides references for further reading.

A model of the IMS development life cycle is presented in Figure 3. Although it resembles the traditional waterfall diagram, this schematic shows the process as a series of transformations between environments. The underlying concept is that we must first understand each operational environment before we can develop the tools needed to support the transformation of knowledge from one environment to another.

The application environment is the arena in which the system will be used. The needs of this environment establish what is to be done; its procedures are generally altered by the delivered IMS. (This is a combination of the Heisenberg Uncertainty Principle and a fundamental restructuring of the environment.)

The analyst's description of what the IMS is to do is created in the development environment. The description usually consists of text and diagrams limited to the scope of the IMS. Part

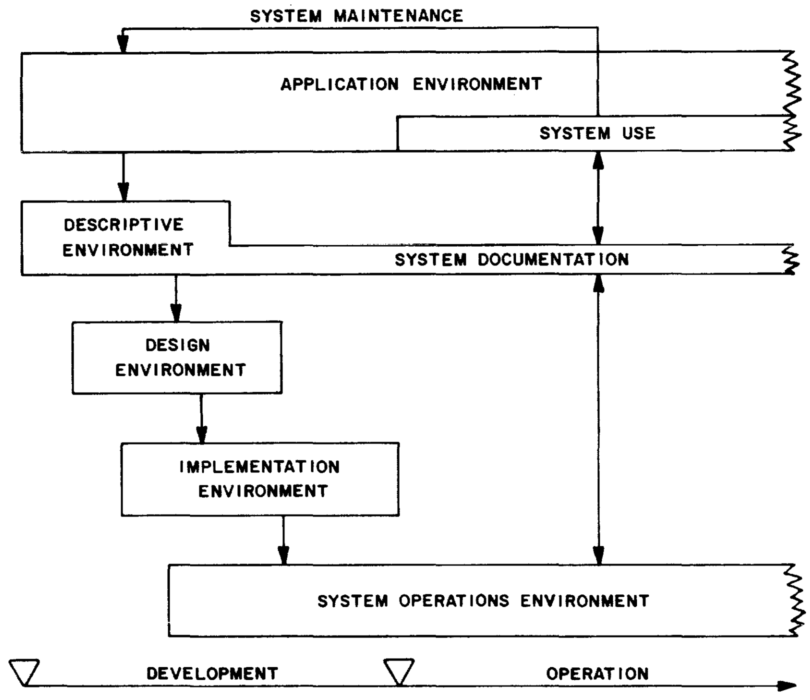


Figure 3—A model of the system life cycle

of the description is used only during the development phase; other parts (e.g., user manuals) are required throughout the life cycle.

In the design environment there is a translation of the description that tells *what* is to be done into a specification of *how* it is to be done. The form may be either text or a formal specification language. This specification is translated into instructions that can be executed. The process is an expansion of the design into code (commonly called programming). This function is carried out in the implementation environment. It is supported by the system operation environment that represents the hardware and software (e.g., computer and operating system) used for testing and the operation of the finished application system. Not shown in the figure are the necessary feedback loops or overlaps due to phased implementation.

In this development model there are five major transformations. The first is from the application to the description. It is independent of programming language, operating system, and computer equipment. The description is typically documented as text that is partially maintained throughout the system's life cycle. The next transformation is from the descriptive text to the design specification. If a formal specification language is used, this is a translation from a text document to an unambiguous description of how the system shall be implemented. (Some software engineering documents limit the role of formal specifications to the requirements that simply describe *what* is to be done. It is now generally recognized that it is impossible to always separate structure from behavior; thus this convention is not used with TEDIUM.)

The design specifications are based only upon the description and are independent of the application environment. They also are generally independent of the implementation language and target system. The next two transformations involve the conversion of the design to an implementable product and a test of that product in a system operations environment. For the target applications, there is little technical risk in these transformations. The major concern is that the design specifications satisfy the application environment needs.

The final transformation is system maintenance. System maintenance consists of (1) correction of errors and (2) changes and expansions to the system. In either case the process involves each activity required during the initial design. For system maintenance, however, the problem may be complicated by the fact that key analysts and documentation are not available. The model shows system maintenance as a feedback loop that restarts the cycle. (This flow is described in greater detail by Blum.[5])

TEDIUM uses this development model to structure the design process. It begins with a statement of requirements that is developed within the application environment. These requirements are produced by the users and application designer. Though in some cases diagrams may be used, the requirements are normally text descriptions. Where the requirements are well understood, they are written in document format; for less well-understood requirements, an outline frequently suffices. The process of requirement definition is intended to document mutual understandings and to work toward the establishment of the users' true needs. The system requirements, therefore, do not act as a fixed document that initiates the design activity (i.e., a "build to" specification). Rather, the requirements represent an initial step in a continuing user-designer dialogue. (The overall process is called system sculpture and is further described by Blum and Houghton.[6])

Once the requirements for an application (or build) are established, the requirements are decomposed into processes and data groups. There is no algorithm for the allocation of requirements to processes and data; it is a highly subjective and personal activity. Of course, the general principles of structured analysis apply. One requirement may initiate many processes (data groups); one process (data group) may satisfy many requirements. Following the examples of Jackson and Warnier, we have made the notation for processes and data symmetric. Depending upon the application, the designer initially may elect to concentrate on processes or data groups. Both processes and data groups are documented as text. Again, outlines may be used where uncertainty exists.

Although there is freedom to work within the design methodology, the final design of an application must reach equilibrium and completeness. This is accomplished when all requirements are satisfied by at least one process or data group, all processes and data groups satisfy at least one requirement, and a data flow exists that includes each process and data group. And finally, the requirements, processes, and data groups must have been organized in hierarchical structures (outlines).

The activity just described represents the transformation from the application environment to the description environment. TEDIUM supports this process by managing the hierarchy nodes and text of the requirements, processes, and data groups. It also provides links between the various entities and thus controls the descriptive network. Text recorded in the descriptive environment may be flagged with respect to importance (great, moderate, detailed) and potential audience (general, user, programmer, operator). TEDIUM provides tools to extract and organize text based upon class, importance, and audience in order to produce general documents and manuals. Thus, the requirements and design data are seen as an initial step in a continuing process of refinement that produces the final application documents. (The documentation process and data structures used to support it are presented in greater detail by Blum.[7])

Once the descriptive environment material is available, the transformation to the design environment begins. Processes are decomposed into executable entities (programs) and data groups are decomposed as relations (tables). Again, programs and tables are linked to the processes and data groups that spawned them. This many-to-many map is also maintained by TEDIUM. It facilitates maintenance by directly linking any program or table with the system elements or requirements that it affects. Of course, the validity of these links depends on the judgment and dedication of the application designer. Though the network can get out of date, the relative ease of update and the ability to produce cross-references to highlight inconsistencies represent improvements over traditional documentation methods.

Figure 4 summarizes the linkage of the various descriptive

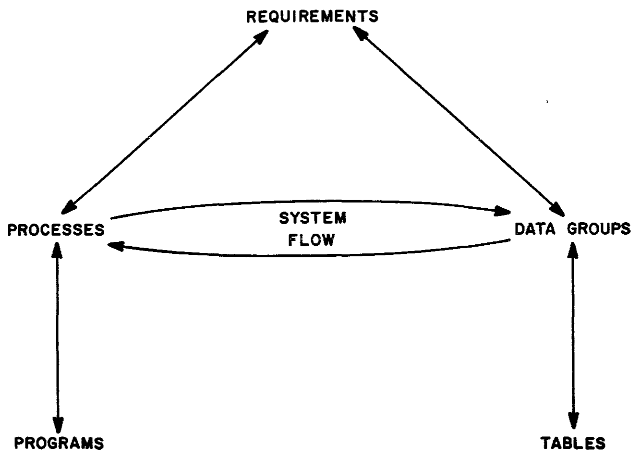


Figure 4—System definition network

and design components. Several points should be noted. First, the documentation process is not mandatory. One can define programs and tables without starting with requirements. For small, limited-use applications, written requirements are not prepared. Second, the definition network allows the designer to conceive of the system as a network with neither top nor bottom, even though processes and data groups are represented as a hierarchy—that is, data flow is possible between data groups at one level of depth and processes at another. Consequently, data flow diagrams that impose a system hierarchy upon the data-process interactions may be managed by TEDIUM; however, they are the exception rather than the rule when the TEDIUM design methodology is used. (A more complete discussion of the design methodology with sample descriptive outputs is given by Blum and Brunn.[8])

Once the programs and tables have been identified they can be specified. An underlying axiom of TEDIUM is that it should minimize duplication and provide immediate access to all established application design information. This is most easily illustrated by considering the process of table definition. (Because TEDIUM relies heavily upon user interaction, the term table was selected in place of the more threatening term relation.)

The TEDIUM data model uses tables with multiple, variable-length keys (index elements) and variable-length records containing zero or more data elements. Unlike a true relational model, it always orders the entries in a table by index element. (This knowledge of access order is essential when one is producing applications for small systems with no DBMS facilities.) The short notation used by TEDIUM is

PATIENT($\underline{\text{PATID}}$) = NAME,AGE,RACE,SEX

where PATIENT is the table name, PATID is an index term, and NAME, AGE, RACE, and SEX are data terms. The underlining of PATID indicates that it is a defined term. That is, the table PATIENT will act as a dictionary for values of PATID. All attempts to enter a value of PATID that is not already in the table PATIENT will be rejected (unless, of course, this feature is overridden within a specification.)

The TEDIUM data model also allows the creation of secondary tables that are subsets of a primary table. For example, the table

PATNAME($\underline{\text{NAME}}$,$\underline{\text{PATID}}$)

is a name index to PATIENT. (PATID is required to provide uniqueness for patients with the same name. There are no data elements.)

The process of defining these two tables can be done in a five-minute interaction session. Once the table definition option is selected, the table name is prompted for. If it has not yet been defined, it is added to the database. Descriptive text information about the table is also requested; this may be supplied now or added at a later date. For new tables, TEDIUM prompts for the index and data elements. Again, if the element has not been defined, it is added to the data dictionary. Element definitions contain an external name (used for prompts), the data type and format, optional validation criteria, and text description. After the data element has been established, the system allows the user to add attributes to the element within this table, that is, defined, mandatory, optional, defaulted to a given value, initialized as a given value, and so on. (A more complete description of this process plus a sample dialogue and printed table listings are contained in another paper.[9])

The TEDIUM schema contains all table and data element definitions. The definitions include both formal parameters and free text. (User requests for information about an element first display the text; a subsequent request produces the formal parameters and validation criteria.) The data types include fixed and variable-length strings, integer and real numbers, date and time. There is also a text data type that calls in the text processor for each input request and the text formatter for all writes. The TEDIUM requirement table, for example, has the following general format:

REQUIRE(APPLICATION,REQUID)
  = REQNAME,REQTEXT

where REQNAME is defined as a variable-length character string of maximum length 120 and REQTEXT is a text data element of arbitrary length.

Programs in TEDIUM are defined by use of minimal specifications. The minimal specification is the least amount of information needed to produce the program within the context of a given system style and the design database. That is, one should be able to give a programmer a minimal specification plus the schema and a style manual and, without any further interaction, receive a complete and correct program that fully satisfies the specification and style requirements. For example, given the system style currently in use at the Johns Hopkins Medical Institutions, the specification for a program that would add, edit, delete, list, change index values, and produce table listings for the table PATIENT and also maintain the consistency between the tables PATIENT and PATNAME would be

Entry program for the table PATIENT

The "Entry" program is an illustration of a generic program. It is a template that accepts specific required and optional commands in order to fully define a program. Another example of a generic specification is that for a program to search through the name index (PATNAME), list out all patients who match a given root name, allow the user to select a patient (or indicate that no patient was found), and return the value of the selected patient's NAME and PATID (or the not-found code). This would be written

```
Prompt program for table PATNAME
  Input is     NAME
  Output is    NAME,PATID
```

In each of these cases the program specification is given as nonprocedural statements. Unfortunately, few applications can be specified exclusively through the use of nonprocedural statements. Thus TEDIUM provides the TEDIUM language either to augment generic specifications or to create complete specifications.

Commands in the TEDIUM language tend to be housekeeping-free and application oriented. For example, in the JHMI style, the PRompt command

```
PR     CONTINUE PROCESSING (Y/N)
```

will print out the string CONTINUE PROCESSING (Y/N) and read an input. If the input is null, it is set to Y. If the input is the escape character, an abnormal program return is made. If the input is a question mark, then a help message is printed. (If none exists, then [NO HELP MESSAGE AVAILABLE] is printed. The user has tools to add or alter help messages after system delivery.) If the input is not Y,N, or any of the above, then the system prints [ERROR] and repeats the prompt. Note how the use of this command assures both completeness and consistency in the final product. (Alternate styles might accept Y,YE,YES,N,NO function keys, etc.)

All TEDIUM commands are written as five fields:

- An optional label
- An optional guard that contains a predicate which must be true if the statement (or block started by the statement) is to be executed
- A mandatory operand
- Parameters for the operand
- Optional control fields that indicate where control should be sent after normal or abnormal execution of operand

This command structure builds a GOTO into each statement. Although this may seem to violate current thinking, it is effective for TEDIUM specifications because

- Specifications are compact and short: normally one CRT screen-full, seldom longer than an $8\frac{1}{2} \times 11''$ printed page.
- All procedural statements are always printed in a box with labels outside the box to the left and control statements outside the box to the right. With this format, control flow and block structure are much easier to recognize than in the more commonly used nested style.

All generic programs are written as expansions of TEDIUM statements and nonprocedural inputs. Nonprocedural statements are used for entry and exit assertions. All generic programs allow the insertion of TEDIUM commands at key points in their flow. Thus one can create a generic program and modify it with custom statements that will alter the flow, function, or interface. A common practice in the TEDIUM environment is to generate a program with the desired functionality quickly and then, based upon designer and user experience with the interactive flow, modify the program; hence the term system sculpture. (A more complete discussion of the specifications, system style, the command language, and the method for creating generic programs may be found elsewhere.[10])

Programs are defined as a program specification and a frame specification. The frame is a generalization of the output environment—interactive or batch, scroll or page, or 80 or 132 characters wide, for example. The frame definition may contain head and foot lines and other specific features which facilitate the maintenance of uniformity within an application. The frame allows programs to process output according to the execution mode. Thus, reports being printed interactively will recognize the end of the screen and prompt to continue or quit. When execution is in the batch mode, the page size is adjusted and the prompt is ignored. Since this is managed by the frame, the same program specification and program generally are used for both interactive and batch reports.

Design is completed when the table schema and program and frame specifications are complete. Since TEDIUM uses the concept of minimal specifications, the translation from the design environment to the implementation and systems operations environment is a mechanical task which requires no knowledge beyond that already available in the design database. TEDIUM takes advantage of this fact by generating programs from the design database. Figure 5 illustrates the flow.

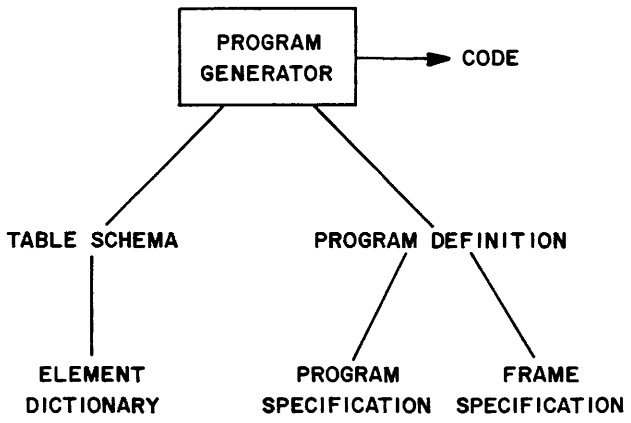


Figure 5—Code generation tree

The generation of programs is a function of program style. Since the TEDIUM specifications are implementation independent, the target language is arbitrary. For example, the flow presented for the PRompt command could be implemented in almost any language. The initial prototype of TEDIUM (called SIMPLE) was designed to produce COBOL outputs. The present version generates MUMPS using a custom coded generator. The next-generation system, currently under development, will have the generator written in TEDIUM and will be designed to produce MUMPS, COBOL, and other target languages. This will provide the tools for users to produce programs that conform to their individual target styles.

The final development process of the life cycle model is maintenance. TEDIUM supports the maintenance process by providing online and printed access to the entire design database. A query capability is available from all specification definition screens. It allows the user to identify programs, tables, frames, and elements, and to examine their contents and structure. Online access is also provided to cross references that indicate, for each program, what programs call and are called by it and what tables are read and written by it. Similar information is available by table name. Program structure listings present a *call tree* for a root program. All of these listings are derived from the design database and are always accurate and timely. Because the TEDIUM programs are written in TEDIUM, it is easy to add new functions to the tool base.

RESULTS

TEDIUM has been in production use since the summer of 1980. There have been 15 or 20 different people using the system. Since 1981 virtually all programming by a staff of eight has been done with TEDIUM. Four TEDIUM users are persons with no previous professional programming experience; after six months of experience, each was able to work independently with users in the design and implementation of applications. The obvious conclusion is that—even though documentation and training materials are severely deficient—the product is understandable at the level of specification and generation. Two other questions remain: (1) can TEDIUM be used to support the development of complex systems as its design objectives intend, and (2) do the features of TEDIUM improve productivity? Each is addressed below.

Scope of Systems Generated

TEDIUM is being used to implement applications that are put into production use. Five major applications have been implemented.

- TEDIUM.—All of TEDIUM except the program generator and selected utility programs are written in TEDIUM. Work on this version of TEDIUM was frozen in September 1981.
- SOCIAL WORK.—A small system used for management by objective, it was the first complete system to be implemented. It has been frozen since June 1981.
- CORE RECORD.—A prototype ambulatory care system was implemented in three clinics, accounting for 75,000 visits a year.[11,12] This version of the system was frozen in December 1981.
- ONCOLOGY CLINICAL INFORMATION SYSTEM (OCIS).—A major tertiary care system[13] with protocol directed patient management,[14] this has been reprogrammed using TEDIUM. Work on system expansion continues.
- OPERATING ROOM SCHEDULING.—This subsystem for the Department of Anesthesiology is in production use; work continues on it.

Table I summarizes the approximate size of each application. It can be seen from these brief descriptions that TEDIUM can be used to produce complex systems.

TABLE I—Overview of TEDIUM-generated applications

Application	Operational Date	Tables	Elements	Programs
TEDIUM	8/80	105	175	318
SOCIAL WORK	3/81	21	78	94
CORE RECORD	3/81	85	245	533
OCIS	6/81	456	1,251	2,177
ANESTHESIOLOGY	1/82	20	66	166

Productivity

Productivity is generally measured in terms of lines of code per unit of work. This measure is imperfect because there are no uniform definitions for a line of code. Moreover, the measure includes only an indication of the bulk of the delivered product. Lines-of-code measures do not address utility of the delivered product (Gladden states that 37% of the systems started were delivered but not used[4]), or the impact of maintenance (generally considered to be at least half of the total life cycle cost.) Software science provides improved measures for volume and effort but does not address the other two variables.

TABLE II—Some productivity measures

		Lines per man-day		Equivalent Custom Code[a]	
Activity	Effort	TEDIUM	Generated[b]	MUMPS	COBOL
Start-up[c]	10 man-years	13.8	101	50	280
Anesthesiology[d]	10 man-days	35.3	247	125	700
Symposium[e]	10 man-days	45.5	402	200	900

[a]Estimated lines of custom code in the target language necessary to produce the same functionality. Based upon analysis by the author.[15] The figures on COBOL are of questionable accuracy but do indicate rough orders of magnitude.
[b]Generated lines are based upon 30 characters per line of MUMPS code.
[c]All programmer activity from the time TEDIUM was first available to the end of 1981. Includes training of new employees, debugging of TEDIUM, etc.
[d]Prototype of the operating room scheduling system. First use of TEDIUM by the analyst.[10]
[e]A symposium program management system developed by the author and a novice.[16]

Table II summarizes three exercises in measuring TEDIUM productivity. More detailed analysis will be initiated when the major system we are working on is frozen. Each system cited is in operation and fully satisfies the users' needs (to the extent that this can be done within resource constraints). Finally, long-term maintenance will be facilitated because

- The size of the specifications is one-seventh the size of the generated programs; thus, there is less to maintain.
- Programs are generated from specifications and the data model definitions, so documentation is always correct and up-to-date.
- TEDIUM's automatic generation of cross-reference indices and set/used tables facilitates maintenance.
- The linkage between programs and user documentation is part of the TEDIUM program.

DISCUSSION

There are very few systematic attempts to address the problems of managing the software life cycle. Most tool development is directed to standalone, nonintegratable packages.[17] Most of the work on specifications ignores the maintenance of a bidirectional link between the design and the delivered product. The computer generally is used as a passive store for design information without any provision of access to the knowledge imbedded in it. Too much of the design and coding effort is spent on the mechanical and repetitive translation of general, predictable requirements into programs—a process that is time-consuming and imperfectly executed. Clearly, there is a need to establish a conceptual framework for the development of automated tools that can manage this activity.

This paper began by outlining a fully automated paradigm for the development life cycle. It continued by illustrating how—for a specialized application area—a prototype environment is approaching this problem. It also suggested that the use of this tool offered an order of magnitude of improvement in productivity, a necessary condition for a fourth-generation development environment. While this specific product is of significant use to a small user community, the key issue is how the concept of a fully automated paradigm will affect the process of software development.

First, we must question whether products such as TEDIUM are transportable. Because TEDIUM represents such a significant change to the established life cycle model and product development flow, there is bound to be strong resistance. The barriers are the existence of program libraries and databases, investments in programmer/analyst training, emotional distrust, competition with proprietary products, variation in local styles and needs, and the presence of backlogs that preclude reassignments from production tasks to training in new methodologies.

The technical barriers are relatively simple to address. TEDIUM currently runs on a minicomputer within partitions of less than 10K. The system can be installed on a microprocessor-based system with four terminals, a printer, and 30M disk for about $20,000. This analyst station could support all development and rapid prototyping and send the generated code to a target machine for compilation and final testing. With the program generator written in TEDIUM, system programmers could adapt the system style to meet local needs or to interface with tools such as a DBMS, distributed processing software, and so on. Thus, if the social and emotional barriers can be overcome, the product is transportable. The growth in popularity of "programmerless" systems demonstrates how flexible the user community is.

Another question of equal importance relates to the marketing of such fundamentally different products. In his review of information systems in the 1980s, Weil notes that, because of a need for stability and support, the number of independent software houses will be limited to a dozen or two out of the universe of hundreds.[18] This implies that, barring a phenomenal success story, leadership in the distribution of fully automated environments will have to come from the established vendors. And each of these already has products that claim to provide fourth-generation functionality.

In conclusion, therefore, we see that there is a user community need for continuity and support and an established vendor need to build on existing product lines. Both these needs reinforce the status quo—hence the resilience of FORTRAN and COBOL. Thus, a fully automated paradigm may find wide acceptance only with fifth-generation computer systems. In the meantime, however, TEDIUM provides an effective tool for high application risk IMS projects. And though the goals of TEDIUM development are limited, it is hoped that our experience will encourage others to consider alternative methodologies for a fourth-generation design environment.

REFERENCES

1. Treleaven, P. C., and I. G. Lima. "Japan's Fifth-Generation Computer Systems." *Computer,* 15 (1982), 8, pp. 79–88.
2. Martin, J. *Application Development Without Programmers.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1982.
3. Focus. 1982 CPU Market Survey, *Datamation,* 28 (1982), 5, pp. 34–47.
4. Gladden, G. R. "Stop the Life-Cycle, I Want to Get Off." *ACM SIGSOFT Software Engineering Notes,* 7 (1982), 2, pp. 35–39.
5. Blum, B. "A Methodology for Information System Production." *Annual Meeting of the Society for General Systems Research on Systems Methodology* (Vol. 26), 1982, pp. 278–281.
6. Blum, B., and R. Houghton, Jr. "Rapid Prototyping of Information Management Systems." *Software Engineering Symposium on Rapid Prototyping,* 1982.
7. Blum, B. "An Approach to Computer Maintained Software Documentation." *NBS FIPS Software Documentation Workshop,* NBS Special Publication 500-94, October 1982, pp. 110–118.
8. Blum, B., and C. Brunn. "Implementing an Appointment System with TEDIUM." *Proceedings of the Fifth Annual Symposium on Computer Applications in Medical Care,* 1981, pp. 172–181.
9. Blum, B. "Program Generation with TEDIUM, An Illustration." National Bureau of Standards, *Trends and Applications 1981,* May 1981, pp. 141–149.
10. Blum, B. "A Tool for Developing Information Systems." In H. J. Schneider and A. I. Wasserman (eds.) *Automated Tools for Information System Design and Development.* North-Holland Publishing Co., 1982, pp. 215–236.
11. McColligan, E., B. Blum, and C. Brunn. "An Automated Core Medical Record System for Ambulatory Care." *SAMS/SCM Joint Conference,* Oct./Nov. 1981, pp. 72–77.
12. McColligan, E., and B. Blum. "Evaluating an Automated Core Medical Record System for Ambulatory Care." *American Medical Informatics Association (AMIA) Congress '82,* May 1982, pp. 206–215.

13. Blum, B. and R. Lenhard, Jr. "An Oncology Clinical Information System." *Computer Magazine,* 12 (1979), 11, pp. 42–50.
14. McColligan, E., B. Blum, R. Lenhard, Jr., and M. Johnson. "The Human Element in Computer Generated Patient Management Plans." *Tenth Annual Conference of the Society for Computer Medicine,* September 1980. Reprinted in *Journal for Medical Systems,* 6 (1982), 3, pp. 265–275.
15. Blum, B. "MUMPS, TEDIUM and Productivity." *MEDCOMP '82,* September 1982, pp. 200–209.
16. Blum, B., and J. Blum. "MUMPS Program Generation Productivity Measures." *Eleventh Annual Meeting of the MUMPS Users' Group,* June 1982, pp. 1–5.
17. Houghton, Jr., R. C. *Software Development Tools,* National Bureau of Standards Special Publication 500-88, March 1982.
18. Weil, U. *Information Systems in the 80's.* Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1982.

A metric of estimation quality

by TOM DEMARCO
Yourdon Inc.
New York, New York

ABSTRACT

This paper reports on a metric of the estimating process called Estimating Quality Factor (EQF). EQF is used to gauge progress in estimating technique and to provide an easily measurable standard for good estimating.

INTRODUCTION

You cannot control what you cannot measure. Anything you do not measure is almost certainly out of control. To say that the estimating process for software is out of control would be relatively kind: In a recent metric survey[1] I conducted, estimated development efforts were plotted against actual efforts, giving the dismal result shown in Figure 1. The caption of Figure 1 may be overly optimistic, because a number of participants dropped out of the survey before completion, and there is every reason to suspect that those who drop out of a productivity survey tend to be among the lower-than-average performers.

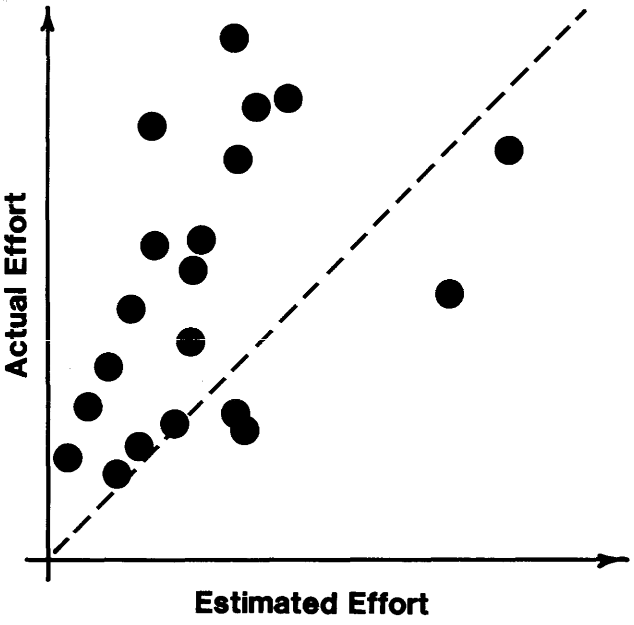


Figure 1—The State of the Art of Software Estimating?

Software cost estimates are notoriously poor. In a recent chemical plant construction project, support software was delivered at a cost nearly 400% of the original estimate; the construction effort was delivered within 9% of its estimate. One very evident difference between the methods used by construction estimators and those used by software estimators is that construction estimators keep rigorous and formal records on the quality of their estimates; software estimators do not. A prevailing software industry standard is to lose estimates the moment they are found to be wrong so that, by the end of a project, no one can say for sure just how bad the original estimates were.

As the software discipline matures, we will need to adopt some of the methods used by estimators in other fields in order to be able to approach their performance. In particular, we will need to formulate a precise and verifiable measure of estimation quality and use that measure to analyze and improve the estimating process. This paper describes one such measure.

ASSESSING THE QUALITY OF AN ESTIMATE

The quality of an estimate must be judged with respect to the duration of the unknown. Estimates generated when management typically requires them—first microsecond of the project—should be expected to have wide tolerances. Later estimates, say from the project 50% point, should have considerably narrower tolerances. Any assessment of estimation quality must, therefore, take into account the point in time at which the estimate was produced.

First Rule: Estimate quality increases monotonically with *accuracy* (how closely the estimate predicts the actual) and with *earliness* (how soon in the project the estimate is produced).

The first, and in many ways most crucial, estimates are commissioned at the very beginning of a project. There are so many unknowns at this point that the tolerance of estimates is unacceptably high. Most projects are begun with estimates that (time will prove) have a tolerance of −20%, +300%. Because tolerances are often not expressly stated, managers assume an expected deviation of −0%, +0%. They deal with the problem of unacceptably wide tolerances by ignoring it. A manager who takes explicit account of the wide tolerances of early estimates, however, is obliged to narrow the risk as the project goes on. This means that he/she must begin to consider estimating as a *continuing process.* A project may begin with the understanding that cost may exceed benefit by a factor of 300%, but it cannot be continued on that basis. Subsequent estimates will have to narrow the unknown so that go/no-go decisions can be made intelligently.

Second Rule: Estimate quality must be assessed not on the initial estimate alone but on the dynamic characteristic of successive estimates and their convergence to actual results.

This means that key project parameters will have to be estimated and re-estimated throughout the project and a formal history maintained. The history might take this form:

ABX Project: Estimated Debugging Effort (Man-Hours)

Date of Estimate	Estimated Value
1/15/80	2,000 hrs
3/15/80	3,500 hrs
6/15/80	5,500 hrs
8/15/80	5,000 hrs
9/15/80	4,750 hrs (actual)

ESTIMATING QUALITY DEFINED

The metric *Estimating Quality Factor* (EQF) is defined by applying the two rules to the estimating history in the following fashion:

EQF = Reciprocal of the average absolute deviation

where the deviation between estimate and actual is expressed as a percentage of the actual. This formulation is more easily understood from a graphic representation of the estimating history, as shown in Figure 2. EQF is now seen to be the area under the actual result ($A \times T$) divided by the shaded area. High *EQF* numbers imply good estimates. The starting point for computation of EQF is taken arbitrarily as the point at which 10% of elapsed time has been used up. Obviously, EQF can be assessed only after the estimated parameter becomes known.

EARLY EXPERIENCE WTIH THE EQF METRIC

For 17 projects studied in Reference 1, average EQF was slightly below 4. The estimating technique used in these projects seems to have been the traditional gut-feel approach. I concluded that an EQF of 4 or lower characterizes the current state of the art of software estimating, and it is a reasonable standard against which to test new estimating techniques. Techniques that raise EQF substantially above 4.0 can be considered improvements.

Experiments in six client companies involved the establishment of politically separated groups of estimating experts according to the method described in Reference 2. These estimating teams did all estimates within their organizations. Project personnel did no estimating and estimators did no development. Estimators were instructed to track EQF for 30 key project parameters and to endeavor to maximize average EQF. They had no incentives beyond this. Explicitly stated, they had no incentive to come up with optimistic estimates, because such estimates would not reflect on their own development skills; no incentives to come up with padded estimates, because estimates that are above the actual reduce EQF just as much as those below; and no disincentive to frequent re-estimation. The estimators applied metric techniques taken from References 2, 3, and 4 as well as from other sources. Average EQF for the experiment improved over the 18 months during which data were collected, from an initial value of 3.8 to 10 or higher.

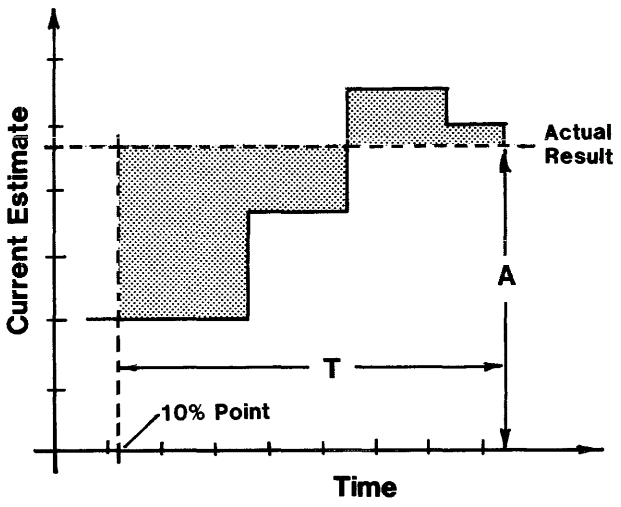


Figure 2—Estimating History (Graphic View)

CONCLUSION

Estimating quality can only improve if we adopt some way to measure and track it. The measure selected should depend on the degree of convergence of estimates to actuals over time. The EQF metric is a palatable metric of estimating quality, at least for some organizations.* Improvement of average EQF in the experiment is the only evidence I know that anyone has ever made progress in learning to estimate software development.

REFERENCES

1. DeMarco, T. "1978–1980 Project Survey Final Report." Yourdon Inc. Technical Report, September 1981.
2. DeMarco, T. *Controlling Software Projects: Management, Measurement and Estimation.* New York: Yourdon Press, 1982.
3. Boehm, B. W. *Software Engineering Economics.* Englewood Cliffs, N.J.: Prentice-Hall, 1981.
4. Putnam, L. H. *Software Cost Estimating and Life Cycle Control: Getting the Software Numbers.* IEEE Catalog No. EHO 165-1. New York: IEEE, 1980.

*One of the participating companies insisted on redefining EQF to weight earlier estimates somewhat higher. Although their formulation was slightly different, it was also a measure of convergence of estimate to actual over time. The specific formula seems to me to be not so important as the idea that there be a predefined formula for estimating quality.

Software productivity measurement

by J. S. COLLOFELLO, S. N. WOODFIELD, and N.E. GIBBS
Computer Science Department
Arizona State University

ABSTRACT

Productivity is a crucial concern for most organizations. This is especially true for software development organizations. Although the term *productivity* is widely used, the difficulty of defining it leads to serious problems in productivity measurement. This paper will attempt to survey some current productivity measures for software development organizations and discuss their deficiencies. A theoretical productivity model that overcomes these deficiencies will also be presented. A practical productivity measure that exceeds current measures by including a quality component will also be described. Although this measure is only a small improvement over contemporary measures, it is a promising step in the direction of better productivity measurement.

INTRODUCTION

Productivity has become a major concern for most organizations attempting to survive in today's competitive market. Productivity improvements are being sought to produce a higher quality product at a lower cost. To assess the effect of steps taken to improve productivity, it is first necessary to have an objective measure. This measure normally assumes the form of a ratio of acceptable products developed to unit of time. Thus, measures such as manufactured components per hour are typical.

Productivity is a major concern for software development organizations. As in other organizations, productivity improvements are being sought to produce higher quality software at reasonable costs. High-level programming languages, software engineering techniques and tools, and new project management methods are all being proposed as ways to improve productivity. For an organization to assess the effects of these new approaches, an objective software development productivity measure is needed. With this type of measure a software development organization can assess its current productivity and determine if and where improvements are needed. Appropriate new techniques can then be applied and a determination made whether productivity was increased by the new techniques.

Productivity measures can also be used by many organizations for comparative purposes. Comparisons may be made of productivity differences between individuals, projects, or competitive organizations. In software development wide variations of productivity are common. Observations of large productivity differences for individuals performing the same task have been cited. When productivity is used to evaluate different software organizations, this comparison must be performed carefully. The reason for this is that although all software developers are producing programs, their products may be no more comparable than transportation manufacturers' products, such as bicycles and aircraft.

Another major use of productivity measures occurs in the product development time and cost estimation phase. An organization must have an estimate of production productivity to project development time and expense adequately. Although software development is often regarded more as research and development rather than manufacturing, costs must nevertheless be estimated. Although still in their infancy, most software cost estimation models require an estimate of the organization's software development productivity as a major input.

Thus, productivity measures are needed for most organizations, including software development organizations. The remainder of this paper will focus on productivity measurement for software development.

STATE-OF-THE-ART PRODUCTIVITY MEASURES

Several different approaches to software productivity measurement are currently used. Historically, the two major approaches are lines of code per programmer month, which expresses productivity in terms of work units; and cost per line of code, which expresses productivity in terms of cost units. These productivity measures can be used to assess the productivity of the entire software development effort. Other productivity measures for specific tasks in the software life cycle can also be used, such as pages of documentation per programmer month and number of test case runs per programmer month.

Another approach to productivity measurement is the software science approach.[1] In this approach lines of code are decomposed into operators and operands in an attempt to produce a more invariant measure than lines of code. Based upon some ratios and counts of unique operators and operands and total numbers of operators and operands, some interesting work and complexity estimates have been formulated.

A new approach to productivity measurement, which abandons the lines-of-code approach completely, is quantifying the functions of programming.[2] This approach concentrates on the number of external user inputs, inquiries, outputs, and master files to be dealt with by the program and uses a weighting approach to calculate a dimensionless number called a *function point.* A variation of this approach is a cost-per-function method, in which a function is defined as a program segment with a single input and output that performs a single transformation or action.[3]

Of the major approaches to productivity measures just discussed, the measure applied most frequently is lines of code per programmer month. This measure as well as the others have several severe deficiencies, which will now be discussed.

DEFICIENCIES OF CURRENT PRODUCTIVITY MEASURES

One of the biggest problems with software productivity is the difficulty of measuring the work accomplished. In other industries, work can easily be measured in terms of products, such as radios produced per unit of time. In software the products are not as easy to quantify. This fact has left the lines-of-code measure as the traditional unit of work. This measure, as well as the software science and functionality types of measures, all suffer from the same problem of not representing adequate work units. Since the difficulty in developing software depends upon the problem being solved, implementation-based metrics such as lines of code, software science size or complexity measures, or function counts will vary with the complexity of the task. Thus, all current software productivity measures fail to take into account the problem size of the task being performed.

Another major deficiency with contemporary measures is the difficulty of extending them into the maintenance phase. One measure for estimating maintenance productivity, reported by Boehm,[4] consists of the average number of instructions that can be modified per man-month of maintenance effort. Modified instructions consist of new instructions or changed instructions. This measure suffers from the same deficiencies of the lines-of-code measure discussed for software development, and it also presents difficulties during maintenance. Since modification of a line of code requires understanding the program, changing it, and accommodating to a possible ripple effect, this measure may be a poor estimate of the amount of work performed.[5]

Another severe problem of all current productivity measures is their failure to incorporate a quality component. The quality of a product must be reflected in productivity measurement. For example, if two organizations achieve the same lines-of-code measure for identical problems, but one organization produces a flawless system and the other is plagued with errors, the software productivity of the organizations should not be regarded as equal. This implies that defect removal costs must be factored into the productivity calculation. In general, other quality factors must also be incorporated into the productivity calculation, and none of them are included in any current measures.

Current productivity measures thus have several deficiencies that have prevented their being reliable or widely used. In the next section a proposed theoretical productivity model will be discussed that overcomes these deficiencies.

PROPOSED PRODUCTIVITY MODEL

For a productivity measure to gain universal acceptance, the problems discussed in the previous section must be resolved. The magnitude of these problems is so great that many feel that the development of such a measure is nearly impossible. Despite the difficulty of solving these problems, it is still essential to identify the characteristics of the ideal solution so that a direction of research can be planned and progress charted toward an eventual solution. It is to attain this objective that a theoretical productivity model is proposed.

The proposed productivity model is calculated as a function of problem size, resources consumed in production, and quality of the product produced. This productivity will be denoted as:

$$P = f\,(PS,\ RC,\ Q)$$

where *PS* is the problem size, *RC* is the resources consumed, and *Q* is the quality of the product.

The problem size component of this productivity measure is meant to correspond to the magnitude and complexity of the task under development. It replaces the lines-of-code and functions measures of work in current productivity measures. This concept is essential to avoiding the current pitfall of productivity measures: rewarding long, inelegant solutions to problems. For example, the current lines-of-code productivity measure would conclude that a programmer who writes a 2000-lines-of-code solution to a problem in 1 month is twice as productive as a programmer who writes a 1000-lines-of-code equivalent solution to the same problem in 1 month. If, instead of lines of code, the problem size is used as a measure of work, the productivity of the two programmers would be regarded as equal, assuming the product was of equal quality.

The problem size for a particular software development effort must be calculated at the requirements level. This is essential, because problem size is independent of implementation size. This notion of problem size rewards cost- and timesaving measures such as reusing code and identifying simple and elegant solutions. The problem size notion is also consistent with the way productivity is measured in other industries. For example, in the automotive industry, productivity is measured by the assembly time of a car; the car corresponds to the problem size. It is not normally measured by the number of welds per hour, since this is an implementation metric. The number of welds per hour is, however, analogous to the lines-of-code measures; and they both significantly affect productivity once a development approach is selected. The key here is that productivity measurement must focus upon the problem size and not upon implementation approaches. The implementation approach is a factor that affects productivity but does not serve as a measure of work in its calculation.

The resources-consumed component of this productivity measure is equivalent to that used in contemporary measures. Thus, this measure may correspond to either a time or a cost calculation of the resources consumed to complete the development of a particular problem size. The time component may be measured in programmer months and the cost component in dollars.

The quality component of this productivity measure recognizes the necessity of evaluating the quality of a product developed. This concept is essential to avoiding the current pitfall of productivity measures: encouraging quantity of code produced without taking into account the reliability of the resulting sofware product. Unless the assumption is made that all software produced is of the same quality, this component must be a part of any productivity measure.

The measurement of software quality is a complex task. In fact, it is impossible to establish a single figure for software quality, since it has many attributes. Some typical software quality attributes are correctness, flexibility, portability, reliability, efficiency, integrity, and maintainability.[6] In a given software development effort, some of these quality attributes may be stressed more than others. Several of the quality attributes, such as efficiency and portability, may also be in conflict with each other. The point here is that a single measure of software quality is unlikely; instead, software quality must be examined in terms of attributes affecting it.

Ths quality component and the problem size component of the proposed productivity measure must be combined to form a measure of the work to be performed. Thus, for a given problem size, the quality desired will determine the actual amount of work to be done. For example, consider a scientific program. If this program were placed on a spacecraft, reliability requirements would cause a larger amount of work to be expended than if the same program did not have these stringent requirements or was used in a different application. Thus, quality and problem size must be combined to form a measure of work in the productivity model.

The proposed productivity model combines these components to form a relationship between work accomplished and resources consumed in the same manner as contemporary productivity measures, but without their deficiencies. Quality is a definite component of this model; therefore, if a product does not meet its quality requirements, the additional rework expenses are factored into the model by adding them to the resources consumed in order to lower the productivity of the developers. Thus, the productivity model does not reward hasty work of poor quality.

It should be noted that this model provides an absolute measure of productivity as a function of work accomplished and resources consumed. This implies all software development organizations can be equally productive, regardless of the product under development. This contrasts greatly with the lines-of-code productivity measures, which expect wide variances of productivity, depending on the application.[7] This absolute nature of the productivity model stems from the fact that work is being measured as a function of problem size and quality. There are many factors, however, that may affect this absolute productivity, such as development tools and techniques, management approaches, and computer resources.[8] These are factors that can be manipulated by an organization in an effort to improve productivity. They are not, however, factors to be included in the productivity calculation.

The proposed productivity model is, of course, currently at a theoretical level. The major difficulty lies in the measurement of problem size and software quality. Once acceptable measures for these two factors are found or reasonably estimated, another problem will be combining these factors into an acceptable measure of work. These are formidable tasks for researchers, with potential spinoffs to many other facets of computer science. The goals of the research, however, are clear; and progress can be charted against these objectives.

PRAGMATIC FIRST STEP TOWARD THE THEORETICAL PRODUCTIVITY MODEL

The proposed productivity model presented in the last section is not usable by current software developers. In this section a pragmatic first step toward the theoretical productivity model calculable today will be described. This productivity measure attempts to go beyond current measures by incorporating a quality component, which will be very simple and will concentrate on the reliability of the software. Other software quality characteristics will be ignored. The productivity measure will also not attempt to calculate a problem size, but instead can use any contemporary measure of work, such as lines of code. Thus, the only significant difference between this productivity measure and current measures is the incorporation of the reliability component. The reliability component will be calculated as the amount of rework time required to fix development errors. This rework time includes both debugging time and the cost of removing the error. Thus, productivity will be calculated as follows:

$$\text{Productivity} = \frac{\text{work accomplished}}{\begin{array}{c}\text{development resources consumed}\\ + \text{ rework resources consumed}\end{array}}$$

The work accomplished can be measured in lines of code or by some measure of functionality currently in use. The resources consumed can be programmer months or any other suitable measure. The rework costs are the resources consumed by correction of errors in the development process. This information is available from the error reports filed against the development effort. This productivity measure penalizes hasty efforts that produce many errors. This productivity measure also supports current software engineering development methodologies that call for extensive reviews throughout the life cycle. These reviews take time, and with current productivity measures the added time detracts from productivity. Under the proposed productivity model, these reviews can increase productivity as long as they are cost effective.

Thus, the proposed productivity measure can be used in an organization to assess its current productivity and measure the effect of development improvement on productivity. This measure can be calculated at the end of the coding phase or at the end of the testing phase to assess the productivity of different life cycle phases. If it is calculated at the end of the coding phase, then error rework costs incurred during the testing and maintenance phases are factored into the productivity equation. If it is calculated at the end of the testing phase, then some interval may be selected, such as the next release, where error rework costs will be calculated.

This productivity measure may also be used (with caution) to compare productivity with that of other organizations. In addition to comparing work measures across organizations (which presents potential problems), error detection and correction mechanisms in the organizations must be examined. They must be examined because the rework component due to errors in the productivity calculation depends on error detection and removal costs. Thus, the productivity of two development organizations producing identical software with identical resources consumed may vary if the error detection and correction tools and techniques of the organizations vary. This forces meaningful productivity comparisons only among organizations with equivalent error detection and correction mechanisms.

Although this proposed productivity measure suffers many of the same problems as other contemporary measures and is limited in scope, the measure does advance the state of productivity measurement to include one facet of software quality. It is hoped that future research efforts will continue to progress toward the development of a complete productivity measure similar to the model proposed in this paper.

FUTURE RESEARCH

We plan to continue our research efforts on improving our pragmatic productivity measure. An attempt is being made to include other quality factors besides reliability in its calculation. The problem size calculation problem is also being addressed. This research has led us into an examination of cost estimation models. Finally, the feasibility of productivity calculation based upon predictive quality metrics, which would enable productivity measurements to take place earlier, is being examined.

REFERENCES

1. Halstead, M. H. *Elements of Software Science.* New York: North-Holland, 1977.
2. Albrecht, A. J. "Measuring Application Development Productivity." In *Proceedings of the Joint Share/Guide/IBM Application Development Symposium,* 1979, pp. 83–92.
3. Crossman, T. D. "Taking the Measure of Programmer Productivity." *Datamation,* 25 (1979), pp. 144–147.
4. Boehm, B. W. *Software Engineering Economics.* Englewood Cliffs, N.J.: Prentice-Hall, 1981, pp. 533–553.
5. Collofello, J. S., and S. S. Yau. "Some Stability Measures for Software Maintenance." *IEEE Transactions on Software Engineering,* Vol. SE-6 (1980), pp. 545-553.
6. Boehm, B. W. *Characteristics of Software Quality.* New York: North-Holland, 1978.
7. Sommerville, I. *Software Engineering.* Reading, Mass.: Addison-Wesley, 1982.
8. Walston, C. E., and C. P. Felix. "A Method of Programming Measurement and Estimation." *IBM Systems Journal,* 10 (1977), pp. 10–29.

The laboratory automation system in the electrical communication laboratories of NTT

by NOBUYOSHI TERASHIMA
Nippon Telegraph & Telephone Public Corporation
Tokyo, Japan

ABSTRACT

Nippon Telegraph & Telephone (NTT) has developed a laboratory automation system to improve the efficiency of research activities in the Electrical Communication Laboratories. NTT has four laboratories at locations distant from each other. However, the research activities each laboratory undertakes are related with what the others are doing. To improve the interchange of information among laboratories, electronic mail, videoconference, and video lecture services have been introduced in the system. It is possible to make inquiries concerning job-processing status and reservations for video conference rooms by telephoning the computer center directly, without going through a human intermediary. Researchers can retrieve information on the newest research results in common with other researchers at any other location. Taking into account the increase in large-scale computing jobs, such as traffic simulation, structural analysis and circuit analysis, newly developed DIPS computers and CAD facilities, such as circuit analysis, have been installed in the system. This paper presents a description of the system's configuration and its characteristics as well as a service outline.

INTRODUCTION

To improve the efficiency of research activities in the Electrical Communication Laboratories, NTT has developed a laboratory automation system with the following services:

1. *Computing.* Computer facilities are available for conversational, remote batch, and local batch processing modes.
2. *Office automation*[1-5] This service consists of research information retrieval, word processing and electronic mail.[6-8]
3. *Design automation.*[9-12] This service provides CAD facilities for LSI logical design, circuit design, and LSI testing for communications equipment.
4. *Speech recognition and synthesis.* This service makes it possible to automatically reserve a video conference room, as well as get information on the operation schedules of the computer centers and job processing statuses, by use of a telephone.
5. Videoconferencing and video lectures.[13, 14] These services make it possible for researchers to communicate with each other from laboratories in widely separated locations.

The present paper describes the configuration and characteristics of this system. It also gives an outline of the services provided by the laboratory automation system.

SYSTEM CONFIGURATION

Figure 1 shows the configuration of the laboratory automation system. The system is composed of DIPS centers, the net-

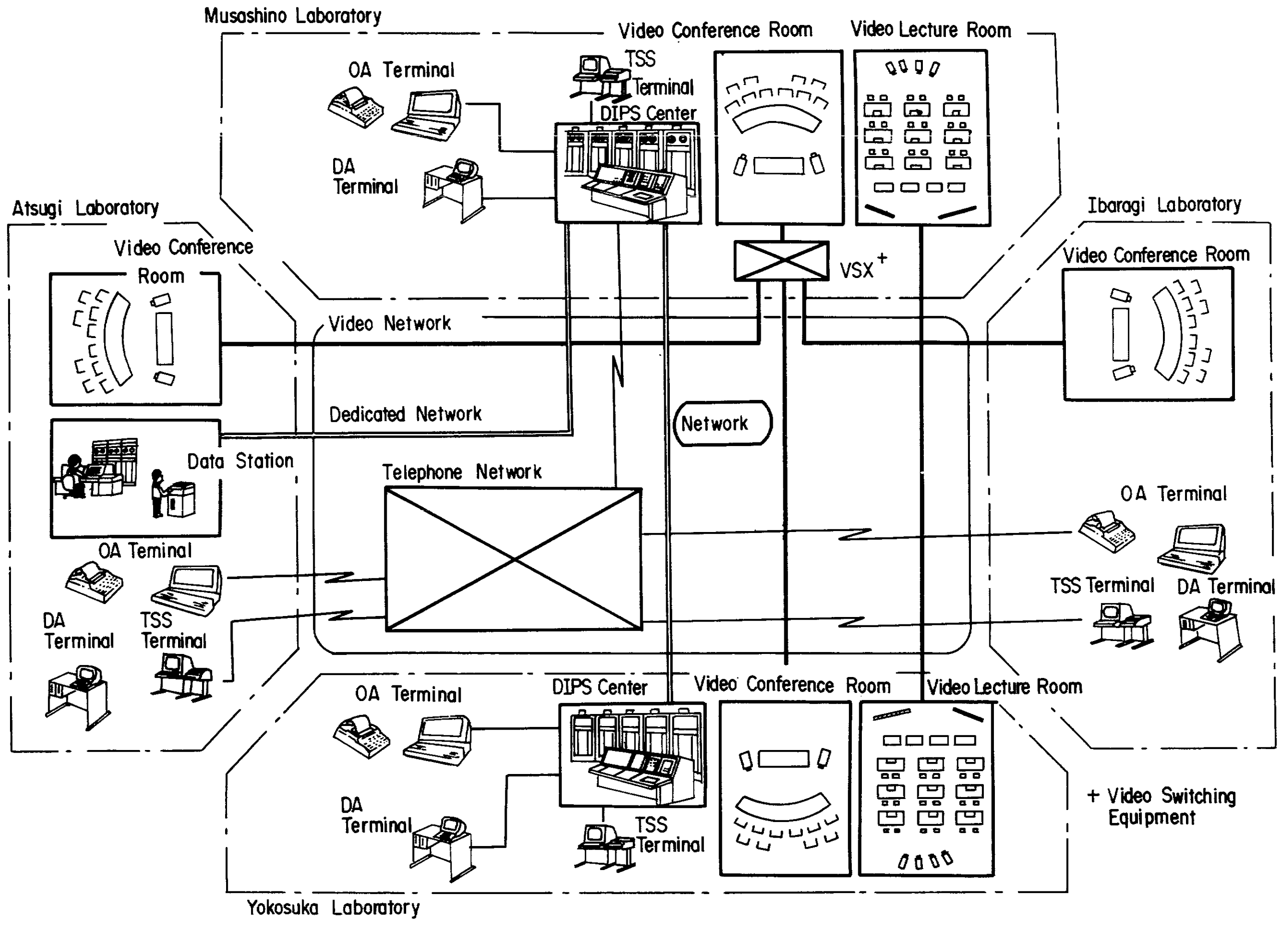


Figure 1—Configuration of the laboratory automation system

work, and various terminals. In addition, a data station is installed at the Atsugi Laboratory.

1. *DIPS centers.* The DIPS centers, where very-large-scale computer systems for DIPS (or Den Den Kosha [Japanese for NTT] Information Processing System) are installed, are located in the Musashino and Yokosuka Laboratories. The DIPS systems are interconnected through a 48 kb/s dedicated network.
2. *The network.* The network is composed of a telephone network, a 48 kb/s dedicated network, and a video network.[15–17]
   a. Data terminals for time sharing, office automation, and design automation, as well as minicomputers and personal computers, are linked through the telephone network.
   b. The 48kb/s dedicated network is used for interconnecting DIPS systems, and for interconnecting the data station in the Atsugi Laboratory with the DIPS system in the Musashino Laboratory.
   c. Videoconferencing terminals are interconnected through the video network, as are video lecture terminals.
3. *Terminals.* Terminal rooms are located in each laboratory for the use of researchers. In the terminal rooms are data terminals, XY plotters, card readers, line printers, and a cassette MT unit. Data terminals and minicomputers are distributed to each research section. They are interconnected to the DIPS centers through the telephone network.
4. *The data station.* The data station is located at the Atsugi Laboratory and is used for transporting design automation programs to the Musashino Laboratory computer center through the 48 kb/s dedicated network.

SYSTEM CHARACTERISTICS

The computer network enables each laboratory to make the best use of computers, programs, and services and to circulate research information efficiently.

To increase computing efficiency, high-speed vector instructions have been implemented in the DIPS systems to speed up the vector operations, which appear quite often in scientific calculations in the laboratories.[18] These instructions are about two times faster than the usual operations. Specifications for the vector instructions are shown in Table I.

To increase office automation and efficiency, basic software and application packages that facilitate Japanese language processing have been developed. Kanji terminals such as the DT-308 and DT-9658 are installed, and high-speed kanji printers, which print documents with Japanese characters at 15 kilo lines/sec, are also implemented at the computer centers.

Additionally, so as to realize the speech recognition and synthesis services, we have developed speech recognition and synthesis techniques we respectively call SPLIT[19] and LSP.[20] The SPLIT method, a speaker-independent speech recognition method, recognizes up to 1000 words of the speech fed directly from the telephone network in real time.

TABLE I—Specifications for vector instructions

No	Vector Instruction		Operation
	Abbreviated Form	Name	
1	VME (VMD)	Vector Move	$Z_i \leftarrow X_i$
2	VECE (VECD)	Vector Elementwise Complement	$Z_i \leftarrow -X_i$
3	VEAE (VEAD)	Vector Elementwise Add	$Z_i \leftarrow X_i + Y_i$
4	VESE (VESD)	Vector Elementwise Subtract	$Z_i \leftarrow X_i - Y_i$
5	VEME (VEMD)	Vector Elementwise Multiply	$Z_i \leftarrow X_i \times Y_i$
6	VEDE (VEDD)	Vector Elementwise Divide	$Z_i \leftarrow X_i \div Y_i$
7	VIPE (VIPD)	Vector Inner Product	$S' \leftarrow S' + \sum_i X_i \times Y_i$
8	VIPCE (VIPCD)	Vector Inner Product with Complement	$S' \leftarrow S' - \sum_i X_i \times Y_i$
9	VCDS	Convert Double to Single	$Z_i[S] \leftarrow X_i[D]$
10	VCSD	Convert Single to Double	$Z_i[D] \leftarrow X_i[S]$

1. Parentheses show double precision operations
2. Xi, Yi, Zi show vector elements
3. S' shows a scalar element

In this method the average speech spectrum is divided into 256 regions. Telephone speech is divided into speech elements of 16 milliseconds each. Then each element is investigated to see which regions are involved. Then the speech is pattern-matched with the patterns of the predefined words. This LSP method is 40% more economical than the PARCOR method.

Figure 2 shows the configuration of the system. In this system, processing is performed as follows.

1. Speech input through the telephone is fed through the circuit control unit and the exchange, where it is recog-

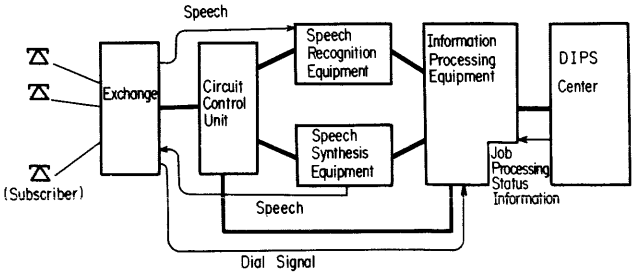

Speech
Speech Recognition Equipment
Information Processing Equipment
DIPS Center
Exchange
Circuit Control Unit
Speech Synthesis Equipment
Job Processing Status Information
(Subscriber)
Speech
Dial Signal

Figure 2—Configuration of speech recognition and synthesis system

nized by the speech recognition equipment. Then the recognized request is stored in the information processing equipment.
2. Information concerning operation schedules and job-processing statuses is sent from the DIPS center to the information processing equipment. Information involving conference room reservations is also stored in the equipment.
3. The recognized request is compared with the information on the operation schedules, job processing statuses, and conference room reservations. Then the answer to the request is generated.
4. The answer is synthesized by speech synthesis equipment and is then sent back to the telephone through the circuit control unit and exchange.

Tables II and III show the specifications for the speech recognition and speech synthesis equipment, respectively.

SERVICE OUTLINE

The services that have been referred to are outlined below in a little more detail.

TABLE II—Specifications for speech recognition equipment

Item	Content
Recognition Object	Isolated word spoken through telephone network
Recognition Method	Speaker independent speech recognition: SPLIT Method
Number of Recognizable Words	1000 words (templates)
Word Length	3.2 seconds
Input Channel	1 channel
Response Time	less than 0.1 seconds
I/O Interface	GPIB (IEEE 488)

TABLE III—Specifications for speech synthesis equipment

Item	Content
Synthesis Method	LSPmethod : a more efficient method than the widely used PARCOR method
Frame Synchronous Period	10 ms
Information	4.8 Kbit/s : information necessary for synthesis of 1 second speech
Buffer Memory	32 Kbytes
Channel	1 channel for input, 1 channel for output
I/O Interface	RS232C (9.6 Kbit/s)

1. *Computing.* Computer power is provided for conversational, remote batch and local batch processing modes.
2. *Office automation.* Office automation involves research information retrieval, word processing, electronic mail, and a library service. The information retrieval service allows access to state-of-the-art technical data, international conference schedules, newly developed products, and so on. The general-purpose word processing service provides facilities for putting together research papers and documents with figures and tables. In the standard-form word processing service, documents used frequently in a fixed form are prepared in the center file. Therefore, these standard-form documents can easily be generated by inputting only a few variable words.
   The electronic mail service allows the sending and receiving of documents through the telephone network. By combining this service and word processing it is possible to create a conference notification document and then deliver it to remote terminals. Specifications for office automation services are shown in Table IV.
3. *Design automation.* LSI design of communications equipment or devices is performed hierarchically in the following manner:
   a. Logical verifications are made at the resistor or gate level.
   b. Circuit analysis is performed at the transistor level.
   Table V shows examples of design automation programs. The LSI design flow is shown in Figure 3.
4. *Speech recognition and synthesis.* The reservation and inquiry system that uses speech recognition and synthesis techniques has been developed to provide the following services:
   a. Videoconference room reservation service.
   b. Inquiry service concerning job-processing status.
   c. Inquiry service regarding the operation schedules for computer centers.
5. *Videoconference and video lecture services.*
   a. Videoconference rooms with video cameras, large-scale video receivers, speakers, and microphones are located in each laboratory. With the use of these rooms, a conference can take place simultaneously at different laboratories.
   b. Video lecture rooms with video cameras, video receivers, microphones, and speakers are presently located at the Musashino and Yokosuka Laboratories. By using these facilities a lecture can be transferred to the video lecture room in the other laboratory, and students in the other laboratory can ask questions and receive answers through the video network.

Specifications for the videoconference and lecture services are shown in Table VI.

CONCLUSION

This paper discusses the basic concept underlying an integrated laboratory automation system that has already been put into practical use at the Electrical Communication Laboratories.

TABLE IV—Specifications of office automation service

No.	Item		Content
1	Information Retrieval	State-of-the-Art Technical Data .	New data with respect to computers, electronic exchanges, transmission systems, LSI, etc.
2	Information Retrieval	Conference Schedules	Information necessary for attending an international conference, such as period, place, deadline for papers, etc.
3	Information Retrieval	Newly Developed Products	Information on newly developed products such as features and prices.
4	Information Retrieval	Schedule Management	Registrations and cancellations of appointments.
5	Information Retrieval	Information Relating to Overseas Business Trips	Names, itineraries and the purposes of people going on overseas business trips
6	Information Retrieval	A Log of Visitors to ECL	Information relating to people who have visited the Electrical Communication Laboratories
7	Word Processing	General Word Processing	(1) Japanese word processing is performed on a non-programmable portable Kanji terminal,the DT-308, using a word processing program called the Kana to Chinese Character Transformation Package. (2) Documents are created by selecting and touching the desired Kana and Chinese characters on the panel of the DT-9658 programmable Kanji terminal. Documents that have been stored in the floppy disk this way can be sent to the CPU upon demand.
8	Word Processing	Standard Form Word Processing	Documents used frequently in a fixed form are prepared in the center file. Thus, standard form documents can easily be generated by inputting a few variable words.
9	Document Commnication	Electronic Mail	Documents can be transferred from one terminal to the other through the network.
10	Library Service		Services include management of books, retrieval of papers, etc. The OCR handscanners are used to input the information with ease.

TABLE V—Specifications of the design automation programs

Design Stage	Program	Function
Functional Design	FDLSIM	• Functional simulation
Logical Design	HIDEMAP	• HSL translation, macro expansion • Data base interface
	TEGAS	• 3 value / 6 value gate level simulation
	LAP - I	• 6 value gate level simulation • MOS Transfer gate simulation
	ADGRAM	• Visible circuit chart generation
	PLACAD	• PLA programming
Circuit Design	LNAP	• Circuit analysis of DC and transient current
	SPICE	• General-purpose simulation program for non-linear DC, non-linear transient and linear AC analysis
Test Data Design	TEGAS	Fault detection simulation

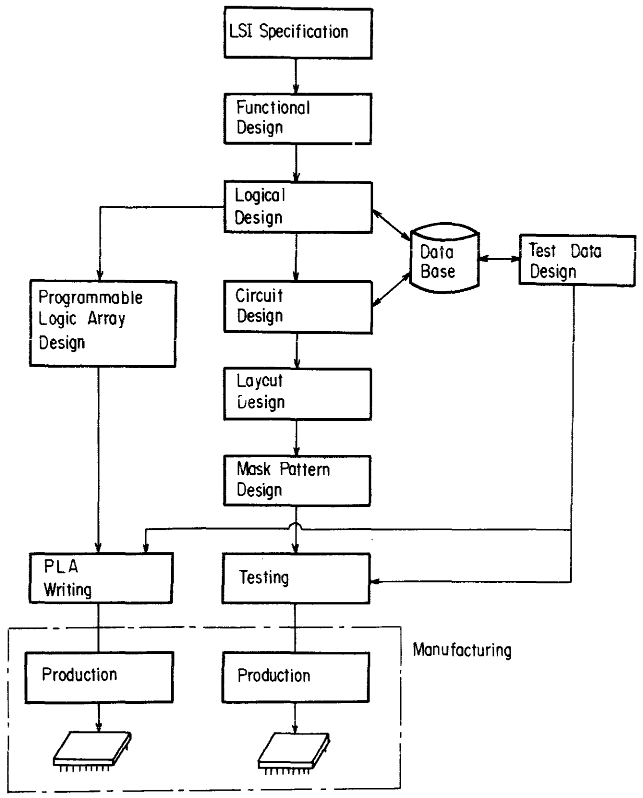


Figure 3—LSI design flow

NTT is now planning to develop the laboratory automation system into a prototype system for the Information Network System (INS). The INS is the advanced public digital network that NTT is now planning to build nationwide.

REFERENCES

1. Wiseman, C. "Integrated Planning for Office Automation." *AFIPS, 1981 Office Automation Conference Digest.* Arlington, Va.: AFIPS Press, 1981, pp. 93–100.
2. Ellis C. A., and G. J. Nutt. "Office Information System and Computer Science," *Computing Surveys,* 12 (1980), pp. 27–60.

TABLE VI—Specifications of videoconference and video lecture systems

Item		Scale	
Video Conference	Monochrome Video	Seating Accommodation	6
		Possible Number of Hookups	up to 3
		Receivers	2
		Cameras	6
		Monitors	2
		Microphones and Speakers	5
	Color Video	Seating Accommodation	10
		Possible Number of Hookups	1
		Receivers	1
		Cameras	1
		Monitors	1
Video Lecture		Seating Accommodation	lecture room=12 auditor room=24
		Possible Number of Hookups	1
		Receivers	4
		Cameras	4
		Microphones and Speakers	1
		Monitors	24

3. "Information Presentation and Manipulation in the Electronic Office." Infortech State of the Art Report—Office Automation (Series 8–Number 3). Maidenhead, England: Infortech Ltd., 1980.
4. "CRT Workstations Envisioned as Office Appliances of Future." *Computerworld,* May 19, 1980.
5. Hammer, M., R. Ilson, T. Anderson, E. J. Gilbert, M. Good, B. Niamir, L. Rosenstein, and S. Schoichet. "Etude: An Integrated Document Processing System." Office Automation Group memo, MIT Laboratory for Computer Science, Massachusetts Institute of Technology, February 1981.
6. Ulrich, W. E. "Introduction to Electronic Mail." *AFIPS, Proceedings of the National Computer Conference* (Vol. 49), 1980, pp. 485–488.
7. Holden, J. B. "Experiences of an Electronic Mail Vendor." *AFIPS, Proceedings of the National Computer Conference* (Vol. 49), 1980, pp. 493–497.
8. Farber, D. J., and S. H. Caine. "Computer Message Systems in the Office." *Proceedings of the 5th International Conference on Computer Communication.* Amsterdam: International Council for Computer Communication, 1980, pp. 43–50.
9. O'Neill, L. A., C. G. Savolaine, T. J. Thompson, J. M. Franke, R. A. Friedenson, E. D. Walsh, P. H. McDonald, J. R. Breiland, and D. S. Evans. "Designers Work Bench—Efficient and Economical Design Aids." *Proceedings of the 16th Design Automation Conference.* New York: IEEE Computer Society, 1979, pp. 185–199.
10. Walsh, M. E. "SPIDER—A Computer Aided Manufacturing Network." *Proceedings of the 14th Design Automation Conference.* New York: IEEE Computer Society, 1977, pp. 431–436.
11. Carley, D. K. "SWESS—The Middle System of a Design Automation Network." *Proceedings of the 14th Design Automation Conference.* New York: IEEE Computer Society, 1977, pp. 425–430.
12. Swiatek, F. E. "A Design Automation System for Telephone Electronic Switching System." *Proceedings of the 14th Design Automation Conference.* New York: IEEE Computer Society, 1977, pp. 419–424.
13. Vallee, J., R. Johansen, and T. Wilson. "Pragmatics and Dynamics of Computer Conferencing: A Summary of Findings from the Forum Project." *Proceedings of the 3rd International Conference on Computer Communication.* Amsterdam: International Council for Computer Communication, 1976, pp. 208–213.
14. Shimamura, K., and Y. Sakasai. "Video Teleconference Systems Psychological Evaluation." *Review of the Electrical Communication Laboratories, N.T.T.,* 29 (1981), pp. 307–328.
15. Clark, D. D., K. T. Pogran, and D. P. Reed. "An Introduction to Local Area Networks." *Proceedings of IEEE,* 66 (1978), pp. 1497–1517.
16. Gravis, H. "Local Networks for the 1980s." *Datamation,* March 1981, pp. 98–104.
17. Breithaupt, A. R. "Project Viperidae, A Bell Labs Computing Network." *IEEE, Digest of Papers of Computer Society Conference,* 1973, pp. 235–237.
18. Terashima, N., T. Ishii, and S. Kojo. "The Firmware Technique to Speed Up the Vector Operations." In *Proceedings of the National Convention Record of the Institute of Electronics and Communication Engineers of Japan,* 1981, p. 174.
19. Furui, S. "A Training Procedure for Isolated Word Recognition Systems." *IEEE Transactions on Acoustics, Speech, Signal Processing,* ASSP-28-2 (1980), pp. 129–136.
20. Sugamura, N., and F. Itakura. "Speech Data Compression by LSP Speech Analysis—Synthesis Technique." *Transactions of the Institute of Electronics and Communication Engineers of Japan,* J64–A (1981), pp. 599–606.

Applications of digital optical disks in library preservation and reference

by WILLIAM R. NUGENT
Library of Congress
Washington, D.C.

ABSTRACT

A Library of Congress pilot project is described in which digital optical disks will be used to provide high-density storage of textual page images. The configuration of the system is shown and the advantages of optical storage for preservation of library materials is explained. It is further shown that while library preservation is the primary objective of the project, the characteristics of digital optical disk provide simultaneous advantages in online reference retrieval. The project plan is described; it calls for scanning and storing one million pages of text over a two-year period. The new system is compared to the library's existing digital optical disk system for demand printing of catalog cards. The future of digital optical disks in libraries is then discussed.

INTRODUCTION

The deterioration of library materials has by now brought about a crisis in all large research libraries. A significant fraction of such libraries' collections of print on paper, over 25 percent by some estimates, are exhibiting the symptoms of terminal acidic decay: turning brown, becoming brittle, and ultimately crumbling into flakes and dust. The cause is well known: the increase in demand for printed matter in the last half of the nineteenth century caused paper manufacturers to make paper from wood pulp and sizing materials that together, by acidic hydrolysis, ultimately form sulfuric acid, which in turn breaks up the normally long and flexible cellulose polymers of the paper. To combat this problem, the Library of Congress has initiated two programs.

The first program, which is proving very successful, is that of gaseous deacidification of large groups of books placed in large vacuum chambers. By this means, books that are in good condition can have their life expectancies extended another 400 to 600 years. A cooperative program with NASA is currently showing the way to achieve deacidification in 5,000-book lots. The second program, the one we address here, will use the high-density digital optical storage of page images as an alternative means of library preservation. A development contract has been awarded to Teknekron Controls Incorporated of Berkeley, California, for a system that will scan and digitize text pages, store them on digital optical disks, and under access control of the library's central computer system will retrieve desired page images for high-resolution display and printout. The digital disk medium itself has a life expectancy of only 10 to 20 years, which is a mere instant in preservationists' thinking; but by the well-established techniques of error detection and correction, early signs of increased soft errors can be detected in time to make a perfect copy on a fresh disk of the same technology or on a future digital medium of a newer technology. By this process, the page images may thus be preserved indefinitely, or for as long as they are believed to be useful.

A second problem of large libraries, of somewhat lesser severity, is that of document delivery to the library patron. With contemporary online computer retrieval systems, such as the Library of Congress' LOCIS system, patrons can find citations of desired items and their call numbers almost instantly. But the time from a patron's request for a work until it is manually searched for in the stacks and delivered can too often exceed an hour. The online accessibility of page-image material on digital optical disks can reduce this document-delivery time to seconds for materials available on optical disks. Greater integrity of the collection is also achieved, avoiding the too common "not-on-shelf" problem.

In the classical library environment, the dual objectives of preservation of materials and providing of frequent public access to them are opposed to each other. Preservation generally means a strictly controlled physical environment, watchful custodial care, and limited public usage. High public usage generally means accelerated wear and deterioration. But page images preserved on digital optical disk can now meet both objectives without conflict, since no wear results from the low-power laser beam reflecting data from the disks.

DENSITY CHARACTERISTICS OF DITIGAL OPTICAL DISK AND PAGE-STORAGE CAPACITY

The primary attribute of digital optical disks is extremely high recording density: Bit areas are generally of the same order of magnitude as the wavelength of the laser light used to record them; that is, in the order of 1 micron or less. Typical numbers for a single-sided 12-inch disk and single-sided 14-inch disk are 10^{10} bits and 5×10^{10} bits, respectively. The general range is within a factor of two of these numbers. The 12-inch Thomson/CSF disk that will be used in the library's application will initially have 0.5×10^{10} user bits, and will later be raised to 10^{10} user bits. We will use the latter number in our library storage examples. On a per-surface basis, the capacity of an optical disk is thus about 150 times that of a magnetic disk.

Although this results in many obvious applications of digital optical disk as a computer-room peripheral for mass storage of conventional character-encoded data, it is this observer's view that the highest potential applications are those involving digitized image storage. For the first time, a practical non-contact medium of extremely high density exists to make such image applications cost-feasible.

THE LIBRARY OF CONGRESS PILOT PROJECT

In order to establish a firm and cost-justified basis for the digital-storage approach to library preservation, the Library of Congress has established a pilot project using digital optical storage technology.

The object of the pilot project is to establish a production environment in which both capital costs and operating costs can be determined, so that decisions on expansion can be made. It was determined that scanning and storing 500,000 pages per year would constitute a project of sufficient size. Other questions to be resolved include the depth of indexing and abstracting required when full text rather than citations only are available. Of high importance as well will be life testing, both in real time and in accelerated time, of the digital optical media and the establishment of test criteria for determining when a disk is beginning to degrade so that replication can be effected before correctable errors become noncorrectable ones. Since library preservation is the primary objective of this program, determining the best procedures to achieve this aim will be a major contribution of this pilot project.

SYSTEM CONFIGURATION

The specified configuration of the Library of Congress digital optical disk system is shown in Figure 1. The core of the system is a Video System Controller that directs all systems input/output activities and provides the functions of video-crosspoint switch and compressor/expander for the digitized images. Two optical disk units are shown, one of which will incorporate a 100-disk jukebox. Two magnetic disks serve as image buffers for materials being accessed by the display clusters and for materials being scanned by either the document-page scanner or the microfiche scanner. The Video Image Controller, a high-speed laser printer (a Xerox 5700), the scanners and a two-terminal cluster will be housed in a computer room in the library's James Madison building and will service two-terminal clusters in three library reading rooms, one in each of the library's three buildings. The initial reading-room configuration will have two user retrieval stations, each consisting of a high-resolution CRT terminal for query entry and for display of full pages of text and an associated medium-speed printer for page-image printout. The Video System Controller is also interfaced to the library's Computer Service Center and uses the library's LOCIS online information-retrieval system for locating citations to page-image material available on digital optical disk.

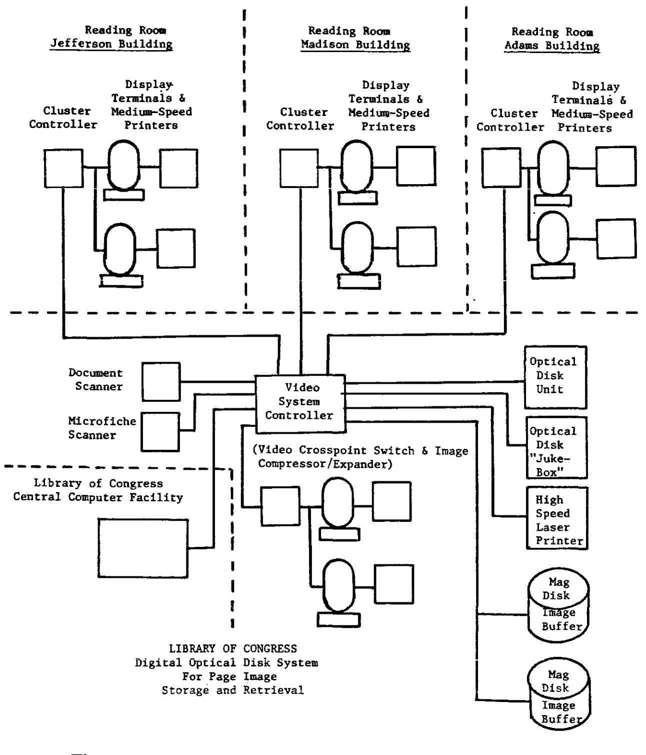

Reading Room Jefferson Building
Display Terminals & Medium-Speed Printers
Cluster Controller
Reading Room Madison Building
Display Terminals & Medium-Speed Printers
Cluster Controller
Reading Room Adams Building
Display Terminals & Medium-Speed Printers
Cluster Controller
Document Scanner
Microfiche Scanner
Video System Controller
(Video Crosspoint Switch & Image Compressor/Expander)
Optical Disk Unit
Optical Disk "Juke-Box"
High Speed Laser Printer
Mag Disk Image Buffer
Mag Disk Image Buffer
Library of Congress Central Computer Facility
LIBRARY OF CONGRESS Digital Optical Disk System For Page Image Storage and Retrieval

Figure 1—The configuration of the digital optical disk system

RELATION TO THE LIBRARY OF CONGRESS'S EXISTING OPTICAL DISK SYSTEM

The library's Cataloging Distribution Service is now using digital optical disk for storage of images of 3″ × 5″ catalog cards in its DEMAND system for high-speed demand printing of catalog cards. This system was developed for the library by Xerox Electro-Optical Systems and was placed in production in August 1982. An exceptionally high resolution system, it laser-scans and laser-reproduces cards at 480 lines per inch. Output is by a specially adapted Xerox 9700 laser printer that produces 12 cards per second. More than 200,000 images of master catalog cards can be stored on one side of the Xerox 14-inch digital optical disk. This is the equivalent of 140 card catalog drawers. The cards output from this system are used to fulfill orders from libraries all over the world, and the DEMAND system has made this operation faster and more cost effective. The success of this first-in-production digital optical disk system has given us confidence that digital optical disk is eminently practical for the storage and retrieval of textual page-images as well as for catalog cards.

THE FUTURE OF DIGITAL OPTICAL DISKS IN LIBRARIES

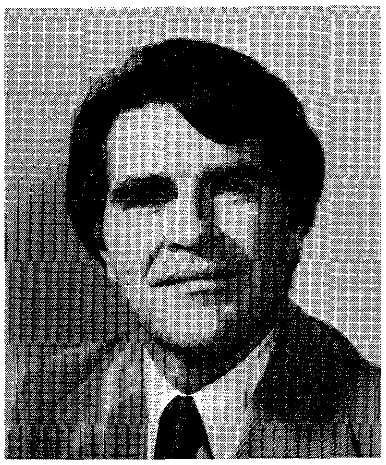
We expect many future applications in libraries as digital optical disk technology is further refined and lowered in cost. In addition to benefits in preservation and reference retrieval, there are significant benefits in storage space and space is an increasing problem in all major research libraries. Today's optical disks offer a shelf-space advantage of about 200X, and we expect a 1000X advantage in a few years. As a second future development, we expect a rapid decrease in the cost of disk players that, in turn, will lead to decentralized use of these devices on a stand-alone basis for special collections, individual reading rooms, and in smaller libraries. The next improvements in greater densities will, in addition, permit the cost-effective storage and retrieval of halftone images and color plates. A particular challenge to be met in the future is that of effectively scanning and encoding pages of mixed material, such as art books that may mix black and white text, halftone graphics and color plates on one page. When this problem is solved, we will have achieved true electronic facsimiles of the printed book.

REFERENCES

1. ADONIS Project Secretary. *ADONIS* Preliminary Draft. Amsterdam: Elsevier Science Publishers, June 1982. 9 pp.
2. Sparks, Peter G. "Mass Deacidification." *Minutes of the One Hundredth Meeting,* Association of Research Libraries, May 1982. pp. 68–83.
3. Nugent, William R. "Optical Disk Technology." *Minutes of the One Hundredth Meeting,* Association of Research Libraries, June 1982.
4. Remington, David G. "Practical Applications at the Cataloging Distribution Service." Minutes of the One Hundredth Meeting, Association of Research Libraries, June 1982. Pp. 80–81.

5. LaBudde, Ed. "Tracking the New Storage Strategy—Optical Disk Memories." *Phototonics Spectra,* May 1982, pp. 61–63.
6. Goldstein, Charles M. "Optical Disk Technology and Information." *Science,* February 12, 1982, pp. 862–868.
7. Brody, Herb. "Materials for Optical Storage: A State-of-the-Art Survey." *Laser Focus,* August 1981, pp. 47–52.
8. McLeod, Jonah. "Optical Disk Storage Technology—New Materials, New Methods." *Optical Spectra,* November 1981, pp. 52–54.
9. Rodriguez, Juan A. "Optical Disk Storage Technology—The Systems Approach." *Optical Spectra,* November 1981, pp. 54–55.
10. Drexler, Jerome. "Optical Disk Storage." Paper presented at Word Processing Industry Conference, Sarasota, Florida, December 3–5, 1980.
11. Bulthuis, Kees, M. G. Carasso, J. P. J. Heemskerk, P. J. Kivits, W. J. Kleuters, and P. Zalm. "Ten Billion Bits on a Disk." *IEEE Spectrum,* August 1979, pp. 26–33.
12. Bartolini, R. A., A. E. Bell, R. E. Flory, M. Lurie, and F. W. Spong. "Optical Disk Systems Emerge." *IEEE Spectrum,* August 1978, pp. 20–28.

PERSONAL COMPUTERS



Tom Fox
Foxware Systems Corporation
Irvine, California

Since their invention over 49 years ago, electronic computing machines have touched the lives of nearly every person on the face of the earth. This has never been more true than now, as we enter the age of personal computers. A personal computer is a dedicated resource, relating one-to-one with its user/master. Whether a production-enhancing tool at one's work, a mind-expanding diversion in the home, or a companion on the road between, a personal computer enriches the lives of those who use them.

For this year's NCC, 24 industry observers and leaders combine in eight sessions to present the 1983 picture of pertinent issues in the personal computer marketplace. Topics range from crucial industry concerns such as software transportability between processors to new-technology methods of protection against program pirates. Included also are practical how-to seminars on business planning with microcomputers and applications for the new generation of mobile carry-along processors. One of the sessions is dedicated to the computer designer's view of the microprocessor that has all but dominated new product introductions since last year's NCC—the 16/32-bit 68000 dual-processor chip.

The most quickly evolving of all computer types deserves special attention to the future, and several of this year's sessions concentrate on the personal computers of tomorrow and their continuing impact on our society. Portia Isaacson has again assembled a panel of experts to look into the future of microcomputing. Industry maker Adam Osborne and seer Jerry Pournelle will project the ways in which personal computers will continue to alter our lives during the rest of this decade.

Software maintenance objectives*

by NED CHAPIN
InfoSci, Inc.
Menlo Park, California

ABSTRACT

Recent survey work indicates that the objectives held by managers of maintenance for application software are partially out of step with the managers' own perceptions of the demands on them. This is a possible cause of maintenance difficulties, to the extent that maintenance managers do guide their behavior by their professed objectives. It is a likely cause of maintenance difficulties if software maintenance is regarded as a service function in an organization, which the survey indicates is the case.

*This paper was presented at the 1982 National Computer Conference, but it was mistakenly omitted from the 1982 *Proceedings* and thus appears in this year's *Proceedings*. It was not presented as part of a 1983 NCC session. AFIPS appreciates Dr. Chapin's contribution to NCC 1982 and regrets the omission of the paper from last year's publication.

INTRODUCTION

The management of software maintenance for application software affects what maintenance is done, when it is done, and how it is done. Software maintenance is managed by supervisors, team leaders, project leaders, programming managers, maintenance managers, and sometimes even by programmers.[4] For convenience and briefness in wording, the term *maintenance manager* is used here for any person, whatever his or her job title, who manages software maintenance.

By *software maintenance* is meant work performed on existing computer programs and systems to correct their performance (for example, to restore to operation after an abort), to adapt them to changed operating environments (for example, to fit with a change in the operating system), to enhance or perfect their performance, to include new or different functions, or to delete functions (for example, to add a new output report).[3] The most important influence on this definition of maintenance is the administrative judgment of how large an enhancement must be to be classified as new development instead of maintenance.[2]

As the total amount of maintenance to be done gradually grows, so also grows the effect of the way that the maintenance work is managed. The swelling backlog of work waiting to be done, the relatively high cost of maintenance, and the lag from request to completion, have all been pointed to as evidence of increasing problems in managing maintenance.[2–5]

As part of a study of the causes of maintenance problems, 15 managers from 12 organizations were surveyed about the demands on them and their objectives in managing maintenance work. The report that follows describes the character of the response, and discusses its implications. This study is a companion to one reported previously at the 1981 National Computer Conference.[1]

RESPONSES

The respondents (maintenance managers) were asked two questions. The first question was what their objectives were in managing maintenance work. Later on the same day, a second question was put: What were the demands made on them in managing maintenance work? Both questions sought open-ended responses and gave the respondents freedom to frame, organize, and order their responses as they saw fit. Essays were not expected but were acceptable, and multiple-choice and true/false responses were neither asked nor encouraged.

In making their responses, the managers had the opportunity to discuss their possible responses with one or two other maintenance managers, and they were encouraged to do so. The responses were also sometimes orally commented on by still other maintenance managers; the respondent was allowed to reply orally but not to change the response. In general, nearly all maintenance managers spoke of their jobs just as they saw them, and were respected for that by the other maintenance managers.

The responses fell predominantly into five categories, but with some notable exceptions as described later. Which category a response fell into depended partly on how the response was expressed, partly on its implications. Some responses fell into more than one category.

The first category focused on the managers' supervisors. The objectives here were to secure recognition of accomplishment. The demands seen from this source were to use resources well and make do with existing conditions.

A second category focused on the organization. The objectives of maintenance managers here were to exercise control to keep systems running. The demands seen from this source involved schedules and budgets, and keeping up communications.

A third category of response focused on the user community. The objectives here were aimed at keeping the user satisfied or even happy. The demands seen from this source mostly concerned how maintenance service was provided.

A fourth category focused on the managers' subordinates. The objectives here dealt with providing acceptable work conditions. The demands involved morale.

A fifth category focused on closely related organizational units, such as computer operations, systems, and procedures, EDP auditing, and data entry. The objectives here dealt with avoiding difficulties. The demands seen involved performance aspects of the software.

Table I lists some representative objectives, and Table II lists some representative demands. For convenience in reading, the table entries shown have been classified by category. Note that the numbering assigned to the categories is arbitrary, though reflective of general management thinking. The ordering of the items within each category, however, is totally without significance and entirely arbitrary.

FINDINGS

The findings here, as in many other studies, depend in part on the manner of collecting the responses. The objectives and demands were separately solicited without first pointing out to the respondents any connection or relationship between them. Both sets of responses were solicited from the same respondents in the context of examining management practices affecting maintenance.

TABLE I—Representative held objectives identified by maintenance managers

Category 1. Toward supervisors
Do maintenance effectively
Reduce costs for work accomplished
Get recognition for job done
Use resources effectively
Keep within estimates
Category 2. Toward the organization
Keep systems running
Act to fit plan, do not react
Keep control over maintenance
Category 3. Toward the user community
Improve communication with users
Keep users happy
Provide good estimates promptly
Satisfy users' requests
Provide a smooth transition from development
Get recognition for a good job done
Increase users' involvement
Reduce problem areas
Make changes in a timely fashion
Prevent users from moving the target
Category 4. Toward subordinates
Improve morale
Keep people motivated
Provide growth potential
Make maintenance work desirable
Keep subordinates happy
Provide training
Category 5. Toward related organizational units
Use less computer time
Do less testing
Satisfy EDP audit requirements
Analyze production problems
Unclassified

TABLE II—Representative seen demands identified by maintenance managers

Category 1. From supervisors
Resolve conflicting demands
Use available resources effectively
Build motivation for staff
Reallocate resources to fit changed workload
Put up with vague directions
Category 2. From the organization
Meet deadlines
Respect budgets
Fill out reports
Participate in meetings
Adhere to organization's standards
Take prompt action on abnormal terminations
Category 3. From user community
Provide immediate response
Provide unreasonable amount of testing
Reduce maintenance backlog
Give quick and accurate estimates
Provide assurance that software works
Reduce time on unproductive work
Accept changes in priorities
Justify estimates
Provide what is asked
Perform within estimates
Category 4. From subordinates
Improve morale
Category 5. From related organizational units
Make the work fit expectations
Unclassified

Before being asked for held objectives and seen demands, the respondents were each asked to what extent they believed that in their organizations the unit charged with doing maintenance work on application software was a "service unit." All of the respondents indicated that they regarded themselves as working in service units. By a "service unit" was meant a unit whose mission, function, and task is to support the functioning of other organizational units of the same parent organization—not, for example, to operate as a profit center selling software maintenance in the open market to any customer. Most maintenance units (but not all) also had software development responsibilities.

The findings from the survey showed much variation in the degree of correspondence between managers' held objectives and seen demands. Some managers displayed a close match. Others displayed large discrepancies. The results reported here are for the aggregate; hence they are not necessarily fully applicable as a picture of the way any one particular maintenance manager sees himself or herself.

An interesting aspect of the findings was the scant attention most maintenance managers gave to ordering either their held objectives or seen demands. Apparently, they do not usually think of them in terms of a hierarchy of priorities. Rather, maintenance managers think of them as associated, sometimes causally; hence the managers take a linear or sequential view of them. Thinking of one objective or demand reminds them of another, which in turn calls to mind still another, and so on, with scant attention to their relative importance.

The most significant finding from the survey was of a mismatch between the managers' objectives, and the demands they saw placed on them. Although partial overlap was present, both the set of held objectives and the set of seen demands covered areas and specific items not covered by the other. While a perfect match was not expected, the overall magnitude and character of the mismatch was unexpected.

This mismatch showed the maintenance manager as having many objectives and no demands in the category of convenience. In fact, this was the largest category of objectives. A mismatch was expected here, but not to the major extent observed. Examples of some of the convenience objectives cited were not to get home late, to have a better qualified staff of subordinates, to consolidate related requests, to have time to test completely, to make maintenance work easier to manage, and to work on a planned rather than a reactive basis. The picture is one of putting self, not the user and not the organization, first.

The user category had the largest number of seen demands, and the second largest number of held objectives expressed. The tenor of the demands was that the users seemed to be acting as though they were buyers of services. Yet the objectives the maintenance managers held were only in part those of vendors of services. Rather than being a crisp "provide what he wants, when he wants it, and at an agreed price," the objectives were diffuse and means oriented, including such items as increasing user involvement, providing a smooth transition, reducing problem areas, keeping users happy, and getting recognition. When such patterns occur in lines of work other than computer software, the usual interpretation is that the manager is experiencing difficulty in providing performance, either from management incompetence, inappropriate resources, or inadequate work methods used by subordinates, or from some combination of them. Whether this interpretation also applies to software maintenance, or not, is unknown.

The supervisor category suggests that the maintenance managers picture their supervisors as wanting them to make do with what they have, yet to adjust to changing requirements as well. The managers' held objectives focus on keeping within estimates, reducing costs, and getting recognition. The match for this category is the closest of all the categories, yet it still shows some divergence.

In contrast, the subordinates category is very divergent. The only common seen demand is ill-defined: improve morale. The held objectives were generally vague, such as to keep subordinates happy, to keep them motivated, and to make maintenance desirable. The maintenance manager appears to be out of touch with his or her subordinates. Yet subordinates are the means by which the maintenance service is accomplished.

The organization was seen as above average in the number of seen demands but low in the number of held objectives. The demands were specific and involved administrative procedures or practices, such as filling out reports or meeting deadlines. In contrast, the held objectives were very general, such as keeping the systems running and providing control. Apparently there are feelings of psychological distance, and the maintenance managers do not feel themselves to be part of the organizational team.

Finally, the managers had very few held objectives in the related organizational-units category, yet demands were seen as present. No convincing interpretation has been found yet for this mismatch, other than a reinforcement of the psychological-distance interpretation noted above.

In summary, the message that emerges is that the objectives held by the maintenance managers do not focus closely on the demands that they perceive. Apparently the managers see themselves, except as regards their relations with superiors, as concerned with trying to keep afloat and with side issues rather than directly with providing service to the user community. This is consistent with the findings of another survey.[2]

A hint of confirmation is available from informal conversations with user representatives. About a third of the user organizations contacted complained that they were not being adequately serviced. Their maintenance needs were not being attended to at a level commensurate, they believed, with their financial contribution (implicit or explicit) to paying for the maintenance effort. They believed that they were paying for more service than they were getting; the most frequent culprit they saw was the maintenance managers' "misdirected efforts." It would be interesting to compare the maintenance managers' held objectives with what the users and others in their organizations regard as their claims upon the maintenance managers.

DISCUSSION

The most probable causes of a self-view of the sort just noted are the maintenance managers' choices of objectives, the emphasis they give to achieving them, and the constraints that are beyond their abilities to affect. Let us look at each cause.

The constraints on the managers in the very near term (such as this week) are real and intractable. The code is what it is, the subordinates' skills are what they are, and the computer programs to be run are mostly set. In the face of this fixity, bugs appear unpredictably, changes requiring unpredictable amounts of effort are to be made in the software, and other work may take away from the available staff and computer time, according to the maintenance managers. Many maintenance managers appear to set as their objectives what will let them live with these constraints from day to day, indefinitely.[1,5]

Many of the constraints on the managers in the longer term, say a year from now, are tractable. For example, in a year's time subordinates' skills and work practices can be drastically changed, code can be made more maintainable, and user needs can be in part anticipated and deliberately scheduled for satisfaction—if the maintenance managers so choose. However, informal conversation indicates that many maintenance managers believe themselves to be powerless or their hands to be tied—"Who's going to pay for it?" and "They won't let me do it," they say. In other lines of management work, such views are often regarded as focusing on the problem rather than on the opportunity, a sign of weakness in management. Whether or not this is also true in software maintenance has not been established.

Managers always give more emphasis or attention in their performance to meeting some objectives than to others. The choices managers make reflect personal preferences, personality, and the mix of skills, as well as a conflict between lip-service and performance. It is just human nature for managers to emphasize what they are comfortable with and what they like to do. The element of job security also affects the emphasis too. The old saw about programmers creating job security by documenting nothing and writing opaque code seems also to apply to some maintenance managers. Keeping a big backlog of unsatisfied user demands is apparently seen by a few maintenance managers as not only ensuring job security but also strengthening requests for increases in the manager's pay and status.

Managers' choices of objectives are really statements of the terms in which the managers see their jobs. Managers can express these in any way they like. Managers may elect to ignore nearly all of the demands they see and to concentrate objectives on other matters. Or at the other extreme, managers can match, nearly one-for-one, objectives against demands. Managers set their own objectives, because other objectives—such as those they hear their bosses pushing—are by definition not the managers' objectives but rather just some more demands upon them. The critical factor is the extent to which their behavior is guided by the held objectives. Managers who profess one thing and then do something inconsistent with it usually get unfavorable attention from their supervisors and associates. Whether or not this is also true for maintenance managers has not been established.

CONCLUSION

The survey reported here has highlighted a discrepancy between the objectives and the demands that maintenance managers see as defining their jobs. The reality that the managers live with is the reality they see—the seen demands, and their choices of objectives. In a service organization, the held objectives and the seen demands should match fairly closely, if objectives direct or guide performance. The survey reported here found a mismatch, with differences in the type, number, and quality of the held objectives and those of the seen demands. Under the assumption just noted, this mismatch is a significant cause of difficulty in managing maintenance.

REFERENCES

1. Chapin, Ned. "Productivity in software maintenance," *AFIPS, Proceedings of the National Computer Conference* (Vol. 50), 1981, pp. 349–352.
2. Comptroller General of the United States. *Federal Agencies' Maintenance of Computer Programs: Expensive and Undermanaged.* Report AFMD-81-25, Gaithersburg, MD: U.S. General Accounting Office, 1981. 60 pp.
3. Lientz, Bennet P., and E. Burton Swanson. *Software Maintenance Management.* Reading, Mass.: Addison-Wesley, 1980.
4. McClure, Carma L. *Managing Software Development and Maintenance,* New York: Van Nostrand Reinhold, 1981.
5. Munson, John B. "Software Maintainability: A Practical Concern for Life Cycle Costs." *Proceedings of the 2nd International Computer Software and Application Conference,* 1978, pp. 54–59.

1983 NATIONAL COMPUTER CONFERENCE COMMITTEES

PROGRAM COMMITTEE

Chairman
Allen N. Smith
Atlantic Richfield Company
Los Angeles, CA

Vice-Chairman
Daniel J. Drageset
Atlantic Richfield Company
Los Angeles, CA

Members

Lorette L. Cameron
ARCO Metals
Rolling Meadows, IL

Tom Fox
Foxware Systems Corp.
Irvine, CA

Donald R. Hyde
IBM
San Jose, CA

David L. Holzman
Holzman & Associates, Inc.
Manhattan Beach, CA

Richard Lampman
Hewlett-Packard Company
Cupertino, CA

Ronald S. Lemos
California State University,
Dominguez Hills
Carson, CA

Lance Leventhal
Emulative Systems Co.
San Diego, CA

Bennet P. Lientz
University of California, Los Angeles
Los Angeles, CA

Randy J. Pilc
American Bell, Inc.
Lincroft, NJ

Toni Shetler
TRW
Redondo Beach, CA

Eugene B. Smith
U.S. Department of Agriculture
Beltsville, MD

Todd Ziesing
Ross Systems, Inc.
New York, NY

CONFERENCE STEERING COMMITTEE

General Chairman
Don B. Medley
California State Polytechnic University
Pomona, CA

Vice-Chairman
James J. Keil
Xerox Corporation
El Segundo, CA

Program Chairman
Allen N. Smith
Atlantic Richfield Company
Los Angeles, CA

Professional Development Seminars Chairman
George R. Eggert
DCASR, Chicago
Department of Defense
Chicago, IL

Finance Chairman
Richard B. Blue, Sr.
TRW
Redondo Beach, CA

Operations Chairman
Mary L. Rich
PFS, Inc.
El Segundo, CA

Registration Chairman
Ron Colman
California State University, Northridge
Northridge, CA

Human Resources Chairman
C. E. Dettman
City of Burbank
Burbank, CA

Special Activities Chairman
Harvey Marks
Transaction Technology Inc.
Santa Monica, CA

Communications Chairman
Ted E. Lorber
Cochrane, Chase, Livingston & Co.
Irvine, CA

Advisor
Herbert B. Safford
GTE Data Services, Inc.
Marina Del Rey, CA

NCC Liaison
Jerry L. Koory
The Rand Corporation
Santa Monica, CA

AFIPS Representative
Jim Kroell
AFIPS
Arlington, VA

Secretary
Bill Fowler
The Rand Corporation
Santa Monica, CA

ADVISOR COMMITTEE

Chairman
Herbert B. Safford
GTE Data Services, Inc.
Marina Del Rey, CA

Members
Linda T. Taylor
Gaskell & Taylor Engineering, Inc.
Los Angeles, CA

Barbara McNurlin
Canning Publications
Torrance, CA

OPERATIONS COMMITTEE

Chairman
Mary L. Rich
PFS, Inc.
El Segundo, CA

Members

Cheryl Chapman
Xerox Corporation
El Segundo, CA

E. Lawrence Doyle
HRFA
Mitchellville, MD

Bruce Hamilton
Xerox Corporation
El Segundo, CA

Steve Herold
Ford Aerospace
Costa Mesa, CA

Virginia Lashley
Glendale Community College
San Marino, CA

Sam Lippman
AFIPS
Arlington, VA

Barbara McNurlin
Canning Publications
Torrance, CA

Robert L. White
Bureau of Census
Washington, DC

PIONEER DAY COMMITTEE

Chairman
Robert V. D. Campbell
The MITRE Corporation
Bedford, MA

Members

Peter McKinney
Harvard University
Cambridge, MA

Richard M. Bloch
R. M. Bloch Associates
Newton, MA

Mort Bernstein
System Development Corporation
Santa Monica, CA

William Aspray
Harvard University
Cambridge, MA

I. Bernard Cohen
Harvard University
Cambridge, MA

Robert L. Ashenhurst
University of Chicago
Chicago, IL

Henry S. Tropp
Humboldt State University
Arcata, CA

Gwen Bell
The Computer Museum
Marlboro, MA

J. A. N. Lee
Virginia Polytechnic Institute and State University
Blacksburg, VA

PROFESSIONAL DEVELOPMENT SEMINARS COMMITTEE

Chairman
George R. Eggert
DCASR, Chicago
Department of Defense
Chicago, IL

Members

Evelyn Teed
Data Entry Specialists
Phoenix, AZ

Sonya (Sam) Anderson
T. H. Garner Co., Inc.
Claremont, CA

Newell Moore
Coast Industrial Exchange
Chatsworth, CA

Evelyn Babler
Defense Logistics Agency
Cameron Station, VA

REGISTRATION COMMITTEE

Chairman
Ron Colman
California State University, Northridge
Northridge, CA

Members

Robert L. White
Bureau of Census
Washington, DC

Shirley Taylor
Property Tax Service
Newport Beach, CA

Barbara Mandell
Barbara Mandell & Associates
La Habre, CA

SPECIAL ACTIVITIES COMMITTEE

Chairman
Harvey Marks
Transaction Technology Inc.
Santa Monica, CA

Members

Carol Johnson
U.S. Department of Agriculture
Beltsville, MD

Lora Meise
Comprehensive Software
Los Angeles. CA

Stuart Shaffer
Litton Industries
Woodland Hills, CA

Mike Marks
Transaction Technology Inc.
Santa Monica, CA

Robert V. D. Campbell
The MITRE Corporation
Bedford, MA

NCC '83 SESSION LEADERS

Eevelyn S. Arkush
Index Systems
Cambridge, MA

Gary Arlen
Arlen Communications, Inc.
Bethesda, MD

Robert Ashenhurst
University of Chicago
Chicago, IL

Herbert V. Bertine
American Bell, Inc.
Lincroft, NJ

Naomi Lee Bloom
American Management Systems, Inc.
Arlington, VA

Grady Booch
Consultant
Lakewood, CO

Alfred Bork
University of California, Irvine
Irvine, CA

Louis J. Brocato
U.S. Department of Agriculture
Beltsville, MD

Howard Bromberg
Bromberg Computer Consultants
Mill Valley, CA

J. Stephen Brugler
Telesensory Systems, Inc.
Palo Alto, CA

Lorette Cameron
Atlantic Richfield Co.
Rolling Meadows, IL

Dr. Ned Chapin
InfoSci, Inc.
Menlo Park, CA

David M. Chereb
Getty Oil Company
Venice, CA

I. Bernard Cohen
Harvard University
Cambridge, MA

Steve Cooper
Intel Corporation
Aloha, OR

Dr. Daniel J. Cougar
University of Colorado
Colorado Springs, CO

Thomas B. Cross
Cross Information Company
Boulder, CO

Dr. Carl Davis
Ballistic Missile Defense
Huntsville, AL

Dr. Gordon B. Davis
University of Minnesota
Minneapolis, MN

Larry E. Druffel
Department of Defense
Washington, DC

Philip Evans
Ferox Micro Systems
Arlington, VA

Martha Evans
Illinois Institute of Technology
Chicago, IL

William E. Farley
Lee College
Austin, TX

James L. Flanagan
Bell Laboratories
Murray Hill, NJ

Dr. Jason Frand
University of California, Los Angeles
Los Angeles, CA

Barry Frankel
Applied Data Research, Inc.
Princeton, NJ

Wayne Green
Wayne Green, Inc.
Peterborough, NH

Michael Hammer
Hammer & Company
Cambridge, MA

Hal Hart
TRW
Redondo Beach, CA

Dr. Paula Hawthorn
Lawrence Berkeley Lab
Berkeley, CA

David M. Herstad
Arthur Andersen & Co.
Los Angeles, CA

David Holzman
Holzman & Associates, Inc.
Manhattan Beach, CA

Dr. David G. Hopelain
Jet Propulsion Labs
Pasadena, CA

Portia Isaacson
Future Computing, Inc.
Richardson, TX

Dr. Evan Ivie
Brigham Young University
Provo, Utah

Alyce Jackson
Technology Research Labs, Inc.
Hawthorne, CA

Steven M. Jacobs
TRW
Redondo Beach, CA

Dr. Philip James
Northrop Corporation
Hawthorne, CA

Dr. Walter Karplus
University of California, Los Angeles
Los Angeles, CA

Dr. Steven I. Kartashev
Dynamic Computer Architecture, Inc.
Lincoln, NB

Dr. Svetlana P. Kartashev
University of Nebraska—Lincoln
Lincoln, NB

John King
DMW Group, Inc.
Carmel, CA

Theodore Klein
Boston Systems Group
Boston, MA

Dennis K. Knight
Hitt, Hartwell & Knight
San Diego, CA

Franklin F. Kuo
SRI International
Menlo Park, CA

Dale Kutnick
The Yankee Group
Boston, MA

Dr. Daniel T. Lee
University of Hartford
Hartford, CT

Dr. Bennet P. Lientz
University of California, Los Angeles
Los Angeles, CA

Rita Gail MacAuslan
Sanders Federal Systems Group
Nashua, NH

Larry Martin
Raytheon Computer Services
Wellesley, MA

Kiyoshi Maruyama
IBM
Yorktown Heights, NY

Richard J. Matlack
Infocorp
Cupertino, CA

Addie Mattox
The Mattox Group
Los Angeles, CA

Eleanor G. Maurer
Blue Cross of California
Oakland, CA

Roy Maxion
Xerox Corporation
Palo Alto, CA

N. Dean Meyer
N. Dean Meyer and Associates, Inc.
Ridgefield, CT

Dr. C. Mohan
IBM
San Jose, CA

Paul Nesdore
Datapro Research
Delran, NJ

Dr. Jack M. Nilles
University of Southern California
Los Angeles, CA

J. Michael Nye
Marketing Consultants International
Hagerstown, MD

Paul O'Grady
Micro Focus
Palo Alto, CA

Bob Patterson
Intel Corporation
Aloha, OR

Michael Plesset
Aerospace Corporation
El Segundo, CA

Steve Puthuff
Fortune Systems
San Carlos, CA

Arthur Pyster
TRW
Redondo Beach, CA

Dr. Jock A. Rader
Hughes Aircraft Company
Los Angeles, CA

William E. Riddle
Software Design & Analysis, Inc.
Boulder, CO

Ken Ross
Ross Systems
Palo Alto, CA

John Ruchinkas
University of Southern California
Los Angeles, CA

Jerald Savin
Main, Hurdman
Los Angeles, CA

Omri Serlin
Itom International
Los Altos, CA

Dr. Phil Shaw
IBM
San Jose, CA

George Sonnemann
Nationwide Insurance
Columbus, OH

John Stidd
Molecular Computing
San Jose, CA

Lynn Svenning
University of Southern California
Los Angeles, CA

James C. Taylor
Socio-Technical Design Consultants
Pacific Palisades, CA

Douglas Theis
Aerospace Corporation
El Segundo, CA

John Thompson
Index Systems
Santa Monica, CA

Dr. Rein Turn
California State University, Northridge
Pacific Palisades, CA

Walter Ulrich
Walter E. Ulrich Consulting
Houston, TX

Virginia C. Walker
Department of Energy
Arlington, VA

James F. Ware
Nolan, Norton & Co.
Lexington, MA

Tom Williams
Computer Design Magazine
Sunnyvale, CA

Amy Wohl
Advanced Office Concepts, Inc.
Bala Cynwyd, PA

Eberhard F. Wunderlich
American Bell, Inc.
Lincroft, NJ

Nicholas Zvegintzov
Zvegintzov Associates
Staten Island, NY

NCC '83 REFEREES

Ackermann, J.
Anderson, Harold A.
Appleton, Daniel

Barnes, Ben
Bartoli, Paul D.
Beamer, Gary D.
Bengston, Deanna
Bertine, Herbert V.
Borko, Harold
Bowen, John B.
Brey, Hans
Brown, Thomas C.

Carlsen, Dave
Cerow, Wayne P.
Chapin, Ned
Charp, Sylvia
Cheney, Paul
Cleveland, L. J.
Conery, John S.
Cotterman, W.
Cox, R.

Date, C. J.
Davidson, Ed
Davis, E. W.
Dawson, Peter P.
De, Prabuddha
Decoster, Wallace
Dordick, Herbert S.

Evans, M.

Finfer, Marcia
Finnila, C. A.

Glaseman, Steve
Gleason, T. C.
Goldberg, Jack
Goodlett, James C.
Gopal, Inder

Hart, Hal
Heafner, John F.
Ho, K. D.
Hohn, William C.
Holmes, William M.
Hopwood, Marsha D.
Horning, James Jay
Huber, K. M.

Ingrassia, Frank
Ives, Blake

Jaffe, J.
James, Philip
Jaramillo, Plas
Jenner, Steve
Johns, Daniel

Kartashev, Steven
Kartashev, Svetlana
Konsynski, Benn
Kozdrowicki, E. W.

Lobel, Jerome
Losch, Myles
Love, Hubert H., Jr.

Manny, Ben
Maruyama, Kiyoshi
Maxion, Roy
McAllister, David
McCullough, T. L.
Mears, Brian R.
Medley, Don
Minoli, Daniel

Navathe, Sham
Nezu, Koji
Nielsen, Francis H.
Nudd, Graham R.

Ormancioglu, Levent

Penedo, Maria H.
Pflager, Richard C.
Plesset, Michael
Pramanik, Sakti
Prescott, Bill
Procella, Maria
Pyster, Arthur

Raiser, Coty
Rine, David
Robb, Richard A.
Rogson, Leon
Rosenbaum, Susan L.
Rosenberg, A. E.
Rosenthal, Soren

Sankar, P. V.
Sastry, K. V.
Schiebe, Lowell H.
Schneidewind, Norman F.
Sealy, David
Shaffron, Nancy
Shaw, Phil
Shoquist, Marc
Shu, Nan
Sibley, Edgar H.
Silver, Aaron N.
Simmons, Dick B.
Smith, Allen N.
Smith, Don D.
Spaniol, Roland D.
Stier, L.
Sulg, Madis

Theis, Douglas J.
Timson, George
Tutelman, D. M.

Vallone, Antonio

Walker, Stephen T.
Wang, C. Charles
Wartik, Steven
Wasserman, Anthony
Wetmiller, John R.
Williams, Donald
Wolfson, Seymour J.
Wunderlich, Eberhard F.

Yost, Robert
Young, C. E.
Young, Jeffrey P.
Young, Steve

Zucker, Steven

NCC '83 SPEAKERS AND PANELISTS

Adkisson, James
Adleman, Leonard
Alker, Pauline
Alon, Eli
Alter, Steven
Athey, Thomas

Balzer, Robert
Barksdale, Gerald L., Jr.
Barley, Kathryn F.
Bateman, Joan
Beadles, Robert L.
Bekey, George
Belady, Les
Belz, Frank
Berkman, Samuel
Berney, Carl
Blackwell, Dennis J.
Bloch, Richard M.
Boehm, Barry
Borgioli, Richard
Brandin, David H.
Brantley, Gladys
Brechtlein, Richard
Brooks, Frederick P., Jr.
Brower, Steven
Brown, John Seely
Brugler, J. Stephen
Burstyn, Paris

Casevoy, Nick
Castano, Gregory L.
Chen, Steve
Cohen, Danny
Collier, James C.
Comisco, Charles
Cooper, Dale O.
Corkery, James
Cornwell, Jack
Corson, Alexandra
Cougar, Daniel
Crandall, Vern J.
Cross, Thomas B.

Damron, Ray
Davis, Gordon B.
Dorn, Phil
Doty, Ted
Druffel, Larry
DuVal, Richard S.
Dutton, William

Emma, Frank J.
Engelbart, Douglas C.
Epstein, Robert
Eulenberg, John B.

Evans, Gwil
Everest, Gordon

Feezor, Betty C.
Feigenbaum, Edward
Fiero, Charles
Fischer, Brian
Florio, Mike
Foster, William
Fox, Steven
Fukunaga, Frank
Fuld, Stephen
Furniss, Mary Ann

Gilb, Tim
Godbout, Bill
Goodell, Robert
Greenlee, M. Blake
Grosz, Barbara J.
Gunter, Tom

Hammer, Michael
Harr, John A.
Harris, Larry
Hayes, Michael
Heuston, Dustin
Hipson, Peter
Hitchcock, Michael H.
Hoffman, Lance J.
Holly, James
Holmes, Fenwick
Hopper, Grace
Hunke, Horst
Hutchison, John

Iverson, Kenneth E.

Jackson, Alyce
Jackson, Barbara
James, Philip
Jamison, H.
Jeffries, Neil P.
Johnson, Bonnie M.
Jones, Wendell W., II
Juliussen, Egil

Kahn, Robert E.
Kalow, Samuel Jay
Karplus, Walter J.
Keim, Robert T.
Keller, John H.
Kemler, David
Killdall, Gary
Kimbrough, Steve
Kinne, Harold
Koffler, Richard

Krishnaswimy, Kris
Kruesi, Elizebeth
Krupp, Marguerite
Kunkler, J. Edward
Kutnick, Dale

Lanergan, Robert
Larson, Judy
Leslie, Mark
Lewin, Marsha D.
Loesh, Robert
Lowenthal, Eugene I.
Lupin, Ed

McAndrews, Robert L.
McCalmont, Tom
McDermott, John
McEvoy, Dennis
McShan, Clyde
Manara, James A.
Mankin, Donald
Margerum, Barry
Martin, Roger
Mateosian, Richard
Matlack, Richard J.
Mellen, Roger
Meyer, N. Dean
Michael, George
Missikoff, Michele
Mohan, C.
Mohrman, Allan M.
Moreland, Thomas W.
Morgan, Howard
Morrow, George
Mostow, Jack
Munson, John
Musa, John
Myers, Dan

Nesdore, Paul
Ness, David
Noble, Chris
Norman, James C.

O'Connor, Rod
O'Grady, Paul
Oettinger, Anthony G.
Orbeton, Peter
Osborne, Adam

Paller, Allan
Parker, Donald M. B.
Parker, Marilyn M.
Piestrip, Ann
Plain, Harold
Plumlee, Hugh

Porter, Benjamin S.
Pournelle, Jerry
Powell, Bruce
Puthuff, Steve

Rabin, Richard
Ramos, Jose
Ratliff, Wayne
Ratner, Andy
Redwine, Samuel
Reifer, Donald
Rich, Elaine
Richards, Stewart A.
Rodman, David J.
Rogers, C.H.
Rogow, Bruce J.
Rollander, Tom
Rosen, Benjamin M.
Russell, David

Sargent, Willis
Schweitzer, Norm
Scooros, Ted
Selders, Wim
Selmi, William
Semon, Warren L.
Sennett, Wayne
Serlin, Omri
Skelton, Thomas
Sloma, Richard S.
Sokol, Ellen
Spencer, Bill
Standish, Tim
Steinfield, Charles W.
Stetter, Gertrude
Stidd, John
Strong, Peter F.
Stvett, Thomas
Swanson, John

Tai, Hoi
Talbot, Guy
Taubert, William
Thomasmeyer, William
Tracy, Rex
Turner, Thomas

Umbaugh, Robert E.

Valleni, Robert
Vaught, Tom

Ware, Willis H.
Whalen, Jack
Whitson, Donald Ray
Wick, John D.
Wilkes, Maurice V.
Winslow, Frederick D.
Winter, Charles
Wolford, Dean
Wszolet, Donald

Yeh, Raymond
Young, Ron

Zollman, Dean
Zuboff, Shoshana

AUTHOR INDEX

AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES, INC. (AFIPS)

AFIPS HEADQUARTERS STAFF

OFFICE OF EXECUTIVE DIRECTOR

Executive Director
Paul Raisig

Executive Secretary
Joan Tackett

COMMUNICATIONS DEPARTMENT

Director of Communications
John Gilbert

Communications Coordinator
Dianne Edgar

Communications Support
Leslie Rodier

Facilities Coordinator
Debra Guazzo

Secretary
Patricia Mayo

Receptionist
Tryphene Miller

FINANCE AND ADMINISTRATION

Director of Finance and Administration
Janis P. Gemignani

Administrative Assistant
Vacant

Accountant
Patricia Whiteaker (Acting)

Bookkeeper
Carrol Reid

Accounting Clerk
John Balderson

Registration Manager
Dennis Smoot

Registration Support
Teresa DiMurro

AFIPS PRESS

Director of AFIPS Press
Christopher N. Hoelzel

Fulfillment Administrator
Olive Shilland

Secretary
Sharon Conway

NCC Proceedings Production Editor
Elizabeth G. Emanuel

CONFERENCE DEPARTMENT

Director of Conferences
James H. Kroell

Administrative Assistant
Susan Robinson

Conference Operations Manager
Sam Lippman

Conference Operations Coordinator
Margaret Dyer

Conference Secretary
Wendy Chin

Marketing Manager
Ann-Marie Bartels

Marketing Coordinator
Loretta Keller

Marketing Secretary
Elizabeth Dollison

Exhibit Operations Manager
Larry Jennings

Exhibit Operations Support
Jill Newman

Exhibit Sales Manager
Richard Dobson

Exhibit Sales Coordinator
Katherine Stormont

Exhibit Sales Support
Molly Finney (Acting)