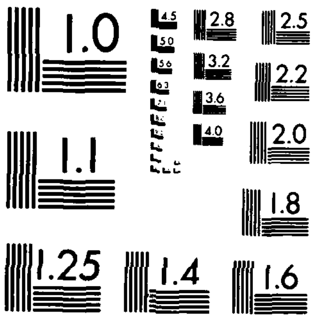
AD-A143 459 LABORATORY FOR COMPUTER SCIENCE PROGRESS REPORT 19 1 JULY 1981-30 JUNE 19..(U) MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE.. M L DERTOUZOS 1/3

UNCLASSIFIED 01 MAY 84 MIT/LCS-PR-19 N00014-75-C-0661 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(12) ⓘ

LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROGRESS REPORT 19

July 1981 - June 1982

Prepared for the

Defense Advanced Research Projects Agency

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER MIT/LCS Progress Report 19	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Laboratory for Computer Science (MIT) Progress Report 19 July 1981 - June 1982		5. TYPE OF REPORT & PERIOD COVERED Annual Progress Report 6. PERFORMING ORG. REPORT NUMBER LCS-PR 19
7. AUTHOR(s) MIT Laboratory for Computer Science Michael L. Dertouzos, Director		8. CONTRACT OR GRANT NUMBER(s) N00014-75-C-0661 DARPA/DOD
9. PERFORMING ORGANIZATION NAME AND ADDRESS MIT Laboratory for Computer Science 545 Technology Square Cambridge, MA 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency Information Processing Techniques Office 1400 Wilson Boulevard, Arlington, VA 22209		12. REPORT DATE May 1, 1984 13. NUMBER OF PAGES 287
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Department of the Navy Information Systems Program Arlington, VA 22217		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computation Structures	Educational Computing	Multiprocessing
Computer Networks	Hardware Systems	Office Automation
Computer Systems	Information Systems	Personal Computers
Computer Languages	Local Networks	Programming Languages
Dataflow	Message Systems	Real-Time
		Specification
		VLSI
		Workstations

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report summarizes the research performed at the MIT Laboratory for Computer Science from July 1, 1981 through June 30, 1982.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

LABORATORY FOR COMPUTER SCIENCE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROGRESS REPORT 19

July 1981 - June 1982

Prepared for the

Defense Advanced Research Projects Agency

Effective date of contract: **1 January 1981**

Contract expiration date: **31 December 1982**

Principal Investigator and Director: **Michael L. Dertouzos**
(617) 253-2145

This research is supported by the Defense Advanced Research Projects Agency under Contract No. N00014-75-C-0661.

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139 (617) 253-5851

C

TABLE OF CONTENTS

→ The Contents include:

ADMINISTRATION

Academic Staff

M. Dertouzos	Director
M. Hammer	Associate Director
A. Meyer	Associate Director

Administrative Staff

J. Badal	Information Specialist
M. Baker	Administrative Assistant
J. Hynes	Administrative Officer
D. Wharen	Assistant Administrative Officer

Support Staff

G. Brown	T. LoDuca
S. Cavallaro	D. Maupin
R. Cinq-Mars	E. Profirio
M. Cummings	P. Vancini

Work reported herein was carried out within the Laboratory for Computer Science, an MIT interdepartmental laboratory. During 1981-82, the principle financal support of the Laboratory has come from the Defense Advanced Research Projects Agency (DARPA), under the Office of Naval Research Contract N0014-75-C-0661. DARPA has been instrumental in the last 19 years and is gratefully acknowledged here.

Final assembly and production of this report was done by Paula Vancini.

INTRODUCTION

The Laboratory for Computer Science (LCS) is an MIT interdepartmental laboratory whose principal goal is research in computer science and engineering.

Founded in 1963 as Project MAC (for Multiple Access Computer and Machine Aided Cognition). the Laboratory developed the Compatible Time-Sharing System (CTSS), one of the first time-shared systems in the world, and Multics -- an improved time-shared system that introduced several new concepts. These two major developments stimulated research activities in the application of on-line computing to such diverse disciplines as engineering, architecture, mathematics, biology, medicine, library science, and management. Since that time, the Laboratory's objectives expanded, leading to research across a broad front of activities that now span four principal areas:

The first such area, entitled Knowledge Based Programs, involves making programs more intelligent by capturing, representing, and using knowledge which is specific to the problem domain. Examples are the use of expert medical knowledge for assistance in diagnosis carried out by the Clinical Decision-Making Research Group; the use of mathematical knowledge for an automated "mathematical assistant" by the Mathlab Research Group; and the use of specific knowledge about budgets for a budget planning system.

Research in the second and largest area, entitled Machines, Languages, and Systems, strives to effect sizable improvements in the ease of utilization and cost effectiveness of computing systems. For example, the Programming Methodology Research Group strives to achieve this broad goal through research in the semantics of geographically distributed systems. Toward the same goal, the Real Time Systems Group is exploring distributed operating systems and the architecture of single-user powerful computers that are interconnected by communication networks. The networks for such distributed environments are studied by the Computer Systems and Communications Group, while distributed file servers and cryptographic protection techniques are pursued by the Computer Systems Structures Group. Other research in this area includes the architecture of very large multiprocessor machines by the Computation Structures and Functional Languages and Architectures research groups, and the use of networks to link large numbers of computers engaged in computationally intensive tasks.

The Laboratory's third principal area of research entitled Theory, involves exploration and development of theoretical foundations in computer science. For example, the Theory of Computation Research Group strives to understand ultimate limits in space and time associated with various classes of algorithms, the semantics of programming languages from both analytical and synthetic viewpoints, the logic of programs, and the links between mathematics and the privacy/authentication of computer-to-computer messages.

The fourth area of Laboratory research, entitled Computers and People, entails societal as well as technical aspects of the interrelationships between people and machines. Examples of research in this area include the use of computers in the educational process by the Educational Computing Group; office automation research carried out by the similarly named Laboratory research group; the use of interconnected computers for planning; as well as the sociological impact of computers on individuals, and the ethical problems of distributed responsibility posed by multiprogrammer systems.

During the past year, the Laboratory consisted of 307 members -- 36 faculty and academic research staff, 22 visitors and visiting faculty, 75 professional and support staff, 90 graduate and 84 undergraduate students -- organized into 16 research groups. The academic affiliation of most of the faculty and students is with the Department of Electrical Engineering and Computer Science. Other academic units represented in the Laboratory membership are Mathematics, Architecture, Humanities, Center for Policy Alternatives, and Sloan School of Management. Laboratory research during 1981-82 was funded by 15 governmental and industrial organizations, of which the Defense Advanced Research Projects Agency of the Department of Defense provided about half of the total research funds.

Technical results of our research in 1981-82 were disseminated through publications in the technical literature, through Technical Reports (TR263-TR276) and through Technical Memoranda (TM199-TM220). The following items are the highlights of the year:

The newly established Educational Computing Group has been augmented with additional people from the MIT Division for Study and Research in Education, notably Dr. Sylvia Weir and her researchers. We are embarking on a major effort in the area of computers and education starting from the results of the pioneering work of Prof. Papert during the last decade. We hope to establish a broadly based effort by pursuing research in the intersection of computer technology, cognitive science, and education with the objective of improving the human educational process.

Another area of emphasis involves our newly created research group on Functional Languages and Architectures. Here, we are pursuing the eventual construction of a new class of computers consisting of hundreds, if not thousands, of interconnected processors, all working toward the same applications goal. Such goals include speech and image understanding, logical inference, and the solution of large, numerically intensive problems, such as weather forecasting. The new opportunity that motivates us to pursue this work is our progressively increasing ability to construct, via VLSI (very large scale integration) techniques, a large number of identical, complex and relatively inexpensive computational structures. Our hope is to develop scalable architectures in the sense that doubling the number of elements in such a system will roughly double the performance under the desired application.

During 1981-82, we have also made substantial progress in our distributed systems research. This major laboratory focus continues to occupy the attention of more than half our people. Our recent results have put us in a position to construct a class of geographically distributed and interconnected systems which strive to balance local autonomy with application cohesiveness. The hardware resources that we designed were successfully transferred to industry and we expect to take delivery of the first commercial-level machines before the end of 1982. These and other machines of the single-user variety are expected to form prototype systems within the laboratory starting in 1983. It is through these prototypes that we plan to implement the collection of research results that we have acquired up to now. In particular, we expect to experiment with languages, operating systems, and applications that establish the feasibility of distributed systems. Such feasibility, in turn, means that an aggregate of arbitrarily many such interconnected and decentralized machines can render at minimum all the functions of a single centralized computer environment -- in the presence of local failures which are likely to be frequent as the number of participating machines becomes large.

During 1981-1982 Drs. David Lebling, Ramesh Patil, Christopher Reeve, and Gerard Vichniac became Research Associates; Dr. Marvin Sirbu joined us as Associate member, and Dr. Sylvia Weir as Principal Research Associate. Finally, Mr. Albert Vezza was appointed Acting Associate Director replacing Prof. Michael Hammer who took a leave of absence for one year.

COMPUTER SYSTEMS AND COMMUNICATIONS

Academic Staff

J.H. Saltzer, Group Leader
D.D. Clark
F.J. Corbató
M.V. Wilkes

Research Staff

J.N. Chiappa
M.B. Greenwald
E.A. Martin
C.M. Novitsky

Graduate Students

R.W. Baldwin
G.H. Cooper
S.R. Curtis
D.L. Estrin
J. Frankel
K. Koile
L.N. Lopez
V. Singh
L. Zhang

Undergraduate Students

L.W. Allen
D.A. Bridgham
M.D. Cunningham
D.C. Feldmeier
D.E. Goldfarb
F.S. Hsu
L.J. Konopelski
C.V. Ludwig
F. Meier zu Sieker
R. Myhill
M.A. Patton
J.L. Romkey
R.S. Teal
D.L. Wilson
K.D. Wright
C.M. Zeitz

Support Staff

D.J. Fagin
N. Lyall
M.F. Webber

1. INTRODUCTION

The Computer Systems and Communication Group of the MIT Laboratory for Computer Science does experimental research on the integration of computer operating systems with data communication networks. Its current projects are in four related areas: alternative local area network technologies, high-performance communication protocols, local network interconnection, and prototype experiments for future portable personal computer terminals. The work of this group is closely coordinated with that of the Computer Systems Structures Group led by Professor David Reed, and somewhat more loosely coordinated with that of the Programming Methodology Group led by Professor Barbara Liskov. Because experimental work in computer systems often requires trying ideas out in realistic application situations, some of the activities of the group mix research with the provision of service facilities to the Laboratory as a whole. The four major project areas are discussed individually in the following sections.

2. NETWORK TECHNOLOGY EXPLORATION

The underlying goal in exploring alternative network technologies is to exploit modern, low-cost digital electronics to provide relatively high-speed data communication among groups of desktop computers and related, specialized, service-providing computers. This basic goal includes the extension of high-speed data communication to personal computers located at home. One such technology, the contention-controlled bus, has been extensively developed by the Xerox Corporation under the name Ethernet. We are comparing the Ethernet with an alternative approach, the ring of digital repeaters using a token for access control. From a theoretical point of view one cannot identify persuasive arguments favoring either one of these technologies over the other. Instead it appears that the primary differences are in things best tested in the field, such as ease of trouble-isolation and repair, frequency of failure, and cost of pre-wiring a building. On each of these issues, there is reason to believe that the ring is a superior strategy.

To gain a better feel for these issues, in 1979 we installed a one megabit/second ring network, a design originally developed by the University of California at Irvine for experiments in distributed computing systems. This ring, which grew to a size of eight nodes was operated side-by-side with a locally-designed variant of the Ethernet, known as Chaosnet. Based on that experience, in 1980 we began work with a subcontractor, Proteon Associates, to develop a 10 megabit/second ring network with a simplified design. This newer ring network, known as the LCS Version 2 ring (and now available from the subcontractor under the trademark ProNet) was the target of a major software support effort of Elizabeth Martin and Larry Allen, after which it began to replace the older ring for some hosts in November 1981. The Version 2 ring has gradually been expanded to connect five nodes. Equipment is currently on order to extend this expansion to eighteen nodes, and plans are being made to increase the number to thirty or so, with the installation of several VAX 11/750 computers for personal computing experiments. Simultaneously, Proteon Associates began commercial delivery of the design, and has completed installation of rings at ten other sites. Host-specific ring interfaces have been designed for computers using the Digital Unibus and Qbus, the Intel Multibus, the S-100 bus, and the LCS nu-bus.

So far, experience with the Version 2 ring in comparison with the Chaosnet and a more recently-installed Xerox experimental Ethernet is largely anecdotal: all these networks work well, they fail to work only rarely, and we believe that the ring is proving easier to repair. To allow us to move from anecdotal to statistical evidence, a network monitoring station is under development. We expect that in the coming year the work in this area will comprise mostly adding stations to the ring and statistics gathering.

As an aside, public interest in ring networks received a strong boost this year with the presentation by the IBM Zurich Research Laboratory of three papers describing a ring network quite similar in design to the LCS Version 2 ring. Although no commitment has been made by IBM to use this approach, a related presentation by members of the IBM Communication Products Division to the IEEE 802 local networks standards committee on the subject of token access rings fueled speculation that IBM may eventually adopt this technology.

A second area of network technology exploration began this year: extension of megabit-per-second communication to computers installed at home. The technology being tried for this project is known in the industry as "broadband", using radio-frequency signalling over coaxial cable with components developed for two-way cable television applications. The long-range goal is to make use of the modern, high-capacity (in channel numbers) two-way CATV systems currently being installed in many large metropolitan areas, including Boston and some of its suburbs. Initially, such communications would be among a few small home computers and larger systems, such as the file-storing hosts at MIT But as the number of home-sited personal computers grows, we envision the arrival of community-located services, such as laser printers and online storage, and an increase of communications among the home computers themselves, to exchange messages, programs, and for multi-player video games. The potential growing demand for such local data communication services has inspired a cooperative venture between our group and the Newton, Massachusetts cable operator, Continental Cablevision. So far this cooperative venture has been limited to discussion of the possibilities and a search for suppliers of suitable hardware.

The hard technical problems in using a residential cable television system seem to be two: accumulation of noise on the channel inbound toward the cable television headend, and assuring no interference with a fully-occupied spectrum of entertainment video materials. The first problem may require the use of digital signal regenerators in place of some of the analog amplifiers in the return path, while the second problem seems to constrain modulation techniques and spectrum shaping (at least on the outbound channel) to closely imitate a standard television signal. An interesting possibility that might tackle both these problems is the use of spread spectrum modulation methods (but present techniques would involve higher cost).

Perhaps harder than the technical problems are policy problems arising from use of systems installed for entertainment video purposes. Our studies on this topic are reported later, in the section on internetwork connection.

Finally, to learn more about the technical aspects of data communication via television cable, cooperation has begun between Ungermann-Bass (a local network supplier), and MIT to allow an Ungermann-Bass subcontractor, Bolt Beranek and Newman, to use two channels of the MIT educational cable television facility as a testbed for broadband equipment. Initial communication experiments began in May 1982, and are expected to continue through the summer and fall.

3. HIGH-PERFORMANCE PROTOCOL STRUCTURES

The second area of research of this group is to understand better the causes of bad performance when computer operating systems are connected to a network. It is by now a common experience to attach a 10 megabit/second local network between two computer systems, fire up some communications software to transfer a file, and observe an effective data rate in the range of 5 to 50 kilobits per second. In explanation of this phenomenon, the words "software overhead" usually appear, but overhead by itself does not really explain a difference between capability and achievement of more than two orders of magnitude -- something more fundamental is wrong. It appears that the primary problem is in the structure of the communication network protocols themselves.

Network protocols, both design and implementation, have evolved until now in a world of telephone lines that are characterized by frequent data errors, point-to-point topology, and most important, a data rate limited to 50 kilobits/second or less -- often only one or two kilobits/second. As one might expect, existing protocol design and implementation experience has been shaped by these properties of telephone lines. Not obvious (until one attaches a 10 megabit/second local network) is just how strong this shaping force has been. Our tentative conclusion from early investigations is that increasing the available data rate by two orders of magnitude so changes the environment that traditional concepts of protocol function, structure, and layers must be largely replaced with lighter-weight approaches. In addition, these lighter-weight approaches must be applied both to the protocol designs and implementations and to the operating system designs and implementations.

One slightly heretical concept that we have explored extensively this year is the complete protocol implementation that is specialized to one application. An example is the file transfer package that in a single small but not very modular program reads data from a file and emits properly-formatted data packets, packets that appear to have been neatly constructed in several layers. Because the only purpose of this package is to transfer files, most of the intermediate protocol layers do not need a full implementation -- only the function required for file transfer need be there. Once this simplification has been made, it is often apparent how to reorganize the remaining functions to combine layers or do things in a different order than the layers might imply. (Karl Wright, in his undergraduate thesis describing such an implementation for the IBM Personal Computer describes this strategy as "cradle-to-grave handling".) The result can be a file transfer application package that is not built out of general-purpose subroutine packages, but that may move files at a rate ten times higher than the usual protocols. Two cradle-to-grave implementations have now been completed, the one already mentioned for the IBM Personal Computer, and a TCP/Telnet by David Clark, for the Xerox Alto.

A closely related idea is the restructuring of a protocol implementation to use implementation modularity that does not conform to the layer boundaries of the protocol. A simple example of this approach shows up in construction of a packet for a multi-layered protocol. Traditionally, an application layer calls an outer protocol layer with a pointer to some data it would like to transmit. The outer protocol layer constructs a packet header, copies the application data into the packet, then calls a lower network layer. This lower layer in turn constructs a header, copies the higher layer's packet into its own data area, and calls on the next lower layer. Only when the packet finally gets to the bottom layer is it discovered that the network is busy and the packet has to be queued anyway. After restructure, this scenario would operate quite differently: the call from the application would drop through all the layers to the bottom, leaving only notes along the way that the client has data to transmit. The next time the network becomes available, a series of calls would go backwards through the layers. The lowest layer would create a header and then call the next higher layer to fill in the data area. This series of calls propagates through the layers, perhaps all the way back to the client, creating a packet with no extra data copies and at a time when it is known that it can be dispatched immediately.

Several experimental implementations using ideas such as the one just described have been carried out this year. Liza Martin developed an IP implementation for UNIX, and Dave Clark developed a byte stream protocol implementation for the

TRIPOS system while visiting the Computer Laboratory of the University of Cambridge. Larry Allen implemented a file transfer package for UNIX that achieves a useful data rate of 130 kilobits/second as compared with 45 kilobits/second for an earlier implementation with more traditional structure. Geoffrey Cooper started work on a Master's thesis on what he calls "soft layering" of protocol implementation. The idea here is that implementation of a protocol layer follow traditional lines but that it be done with knowledge both of the application that the client layer has in mind and also the implementation strategies of lower layers.

A second area of rethinking of organization applies to operating systems themselves. A high-speed communication network places tremendous stresses on the facilities that an operating system provides for coordinating parallel activities. Demands for attention from a network are usually quite unsynchronized with other activities inside the host (as compared with a disk, which usually asks for attention only after you poke it) and successive demands may arrive with only a millisecond headway separating them. More subtly, a decision to queue a packet for transmission (say in acknowledgment of one that just arrived) sometimes needs to be reevaluated if more data becomes available to add to the packet before the transmission actually occurs. Finally, whatever mechanisms are used for sharing data between parallel activities must be designed with the recognition that simple copying of data from place to place consumes an inordinate amount of time.

The folklore in this area says that one should attack these problems by minimizing the number of process scheduling operations. Such a strategy may have the unfortunate side effect of not exploiting some opportunities for parallel activity. That observation in turn leads to the idea that what is really needed is very light-weight parallel activities. We have now completed three experimental implementations of light-weight activities, one for multiple tasks within a process of the UNIX operating system, by Larry Allen; the second for a queue-driven subroutine dispatcher for the PDP-11 MOS operating system, by Noel Chiappa; and the third for a multi-task module for the TRIPOS system by David Clark. The first was the basis for a high performance TCP implementation. The second was the basis for an internet gateway implementation that is still being checked out. (An earlier prototype of this gateway has operated as a link in a chain where end-to-end data rates of over 400 kilobits/second were achieved.) The third was the basis of a revised protocol implementation that achieved a transmission rate improvement of a factor of ten (from 30 kilobits/second to 300 kilobits/second) over the previous implementation.

A third area of organizational rethinking relates to protocols themselves. Professor David Reed of the Computer Systems Structures Group has led the development and implementation of two protocols that reduce the use of acknowledgments to a bare minimum. One is used for rapid transfer of files, the other for rapid transfer of bit maps for displays. It was this latter protocol that operated at the 400 kilobit/second rate mentioned earlier. Another performance-oriented protocol concept is that of source routes: the originator of a packet places a complete set of routing instructions inside the packet so that as the packet proceeds through internetwork gateways it can be forwarded without any computation or table lookup by the gateways. Vineet Singh, in a Master's thesis, developed a plan for a service that calculates source routes for use in such a system. His service allows for hierarchical organization boundaries, with a separate route-calculating service in each organization, yet it retains the essential advantage of high-performance transmission of individual packets.

Although not having exactly the same goals, another project was started in the area of protocol performance: the network attachment of low cost personal computers using the same large-scale protocols as do large mainframes. The goal here is to provide convincing evidence that even a low-performance desk-top computer can participate as a full member of a data communication network. Three IBM Personal Computers were acquired, on the basis that that recently designed machine is typical of the latest wave of 16-bit machines, with an address space large enough and a processor fast enough to allow a full protocol implementation. Since the primary interest in this project was to learn about software feasibility, and it seems likely that manufacturers will soon provide off-the-shelf network hardware for this class of machines, we chose to use as the initial hardware connection just the 9600 bit/second RS-232 asynchronous line connection that comes with the IBM Personal Computer. Several such lines are run to a Digital LSI-11 computer that is to be programmed as an internetwork gateway and attached to one of the higher speed local networks.

Much of the first stage of this project consisted in getting support tools in place to facilitate programming for the IBM Personal Computer. Lack of availability of a native assembler and awkwardness in the mechanics of use of the native compilers for the Personal Computer led to a decision to use a cross-assembler, cross-compiler, and downloading system from our PDP-11/45 UNIX machine, with the result that most of the programs being developed by us for the Personal Computer are in the C language. Two related tools for the Personal Computer were also developed, a terminal emulator and a clock-driven profiler that tells where programs are spending their time. David Bridgham and John Romkey have done most of this work, with both help and guidance from Wayne Gramlich.

The first completed protocol program is the file transfer package using the protocol TFTP/UDP/IP, mentioned earlier. This package achieves a useful data rate of 3500 bits/second over a 9600 bit/second line; most of the difference in these rates can be explained by low performance of the floppy disk software and hardware packages of the Personal Computer. When doing memory to memory file transfer, a data rate of about 12000 bits per second was achieved using a 19200 bit/second line between two Personal Computers. (In both cases the communication line data rate was adjusted to the lowest rate that did not add any bottlenecks.)

Two further protocol implementations are underway for the IBM Personal Computer, a remote login (Telnet/TCP/IP) protocol and an acknowledgment-free file transfer protocol (Blast/UDP/IP).

4. LOCAL NETWORK INTERCONNECTION

The third major research area of this Group concerns interconnection of local area networks. Most laboratories exploring this topic have concerned themselves with only one aspect of this problem, namely that available local network technologies are limited in area to one or a few buildings and limited in connectability to a few hundred nodes. These limitations lead to the conclusion that larger networks must be built up by interconnecting smaller ones with forwarding nodes, called gateways. We have added to this consideration a second aspect that we believe is equally important in its technical impact: administrative boundaries within organizations and between organizations also shape the boundaries of local networks and add management considerations such as policy control (e.g., privacy, authenticity, and accounting) on data flow from one local network to another.

There seem to be two models on which the interconnection of a large number of local networks can be based: concatenation of adjacent local networks, or systematic hierarchical interconnection. With the first model, gateways are placed at opportune points where two or three local networks are adjacent in coverage, and communication from one point to another takes place over as many intervening local networks as necessary. In the second model, a higher-level network called a "spine" or "backbone" is installed, with geographical coverage of the entire community of interest but with attachment only of gateways to the several local networks. With this approach, communication from one point to another goes from a node through its attached local network to a gateway that passes the message to the spine network; it travels across the spine to a second gateway directly into the local network to which the target node is attached.

When administrative boundaries are taken into consideration it is apparent that the model of concatenation of adjacent networks has serious shortcomings. For long distance communication one must depend on correct operation and benevolent administration of all the intervening local networks and gateways. Bandwidth, delay, and reliability of connections are limited by the worst example of each parameter along the path. From the point of view of the manager of one of the local networks, much of the traffic he is carrying may be "tandem" traffic, that is, long distance communication that neither originates nor terminates in his network; this makes control of load, and thus performance, for his own users hard to accomplish. The alternative, hierarchical model, with its backbone network, has corresponding advantages. The backbone network and its gateways can be centrally administered, while the local networks can be locally installed and managed. Responsibility for long-distance communication is clearly defined, which means that trouble isolation, repair, and capacity planning are easier to accomplish.

It is this line of reasoning that has led us to recommend the hierarchical model to the MIT Director of Computing and Telecommunication Resources as the basis for a campus-wide network.

A second area of interest in the area of network interconnection relates to the proliferation of different communication protocols. Unfortunately, the many different sources of protocol implementations are not converging to a single approach. Instead, it appears that strong differences in opinion as to relative importance of different issues is leading to send quite different families of protocol design. The differences stem from application differences, environment differences, and sometimes issues of taste and vendor strategy. These families differ in the same ways (and for the same reasons) that computer programming languages differ. For the moment, at least, any forces for standardization appear to be neutralized by the forces of diversity, which means that every attempt at network interconnection inevitably encounters incompatible communication protocols.

At the lowest level, removing a packet from one local network (say an Ethernet) and placing it on another (say a ring), there is generally not a very serious compatibility problem, because although link-level communication protocols are often very different, they are usually very modular and higher-level protocols are often designed in anticipation of using many different link-level layers and they do not usually depend on having a single link from end to end. The gateway between local networks has a problem that is analogous to that of taking letters from a truck and putting them on a bus or handing them to a letter carrier. One must not ask the letter carrier to walk away with a 100 kg sack of mail, but unless the carrier insists he can handle only letters weighing less than 15 grams there is usually not a serious forwarding problem.

The real difficulties arise at the higher levels of protocol -- the end-to-end transmission protocols and above. After several not-very-satisfying attempts to build protocol-translating gateway programs, we have reached a second important conclusion concerning network interconnections that cross administrative boundaries: Having a protocol translator at the gateway between two networks is about as ineffective as having a long-distance telephone operator act as a language interpreter in a call from Spain to Finland (or, more analogously, automatic translation of Cobol programs into PL/I). That is, there are certain stylized applications for which the approach can be acceptable (for example, remote login and mail forwarding), but as a general communication system it is fundamentally unworkable. The reasons are many, but mostly they boil down to non-comparable semantics in the higher-level protocols. A simple example is translation between two different file transfer protocols, one of which requires that the first fact received be the size of the arriving file while the second allows the sender not to reveal this information until the last packet it sends. To translate between these protocols, a gateway must provide buffer storage large enough to hold the entirety of the largest file that it might be asked to forward. More subtly, since the transmission to the recipient cannot begin until the sender has dispatched the last packet, the sender's protocol may, by timeout, expect commitment of the transaction long before it is clear that the recipient can make that commitment. If the translating gateway returns an early acknowledgment to the sender committing acceptance of the file, then the gateway must honor that commitment with the same degree of reliability as the intended recipient; this commitment means that the gateway must have reliable storage and recovery procedures equal in quality to those of the most sophisticated recipient system.

Even when two different protocols appear superficially similar enough to allow on-the-fly translation, often they are built on different assumptions about network delay, probability of specific kinds of error, or recovery technique; in these cases a translating gateway may have the difficult-to-predict property that it works only when traffic load is light and no errors occur; less-favorable conditions lead to communication breakdown and application failure.

The alternative to translating gateways is end-to-end agreement to use the same protocol. This approach has two implications, one for hosts and the other for gateways that connect networks. For hosts, it means that two different hosts cannot communicate unless they have some common protocol. This implication leads to multiple protocol implementations and to application-specific translation programs. For gateways, it means that packets of several different end-to-end protocols may require forwarding. Since the forwarding strategies of different protocols can be quite different, the gateway must therefore be organized to deal with multiple routing strategies, multiple addressing schemes, multiple error handling and recovery procedures, and multiple examples of local state information.

Realization of this requirement has led us to design and implement a multi-protocol, multi-network gateway package for use in network interconnection. This package, developed by Noel Chiappa, is written using the C language and the MOS operating system, and operates on Digital LSI-11 computers. It has been designed with the ability, at the lower level, to connect to any local or long-haul network; drivers for the LCS Version 1 and Version 2 rings, the Xerox experimental Ethernet, the Chaosnet, the Xerox-DEC-Intel 10 Mbit/second Ethernet, the ARPANET (by Robert Baldwin), and low-speed telephone lines (by David Bridgham) have been developed or are planned. At the higher level, internetwork forwarding packages for IP and Chaos protocols have been implemented, and packages for X.25, DECNET, Xerox NS, and IBM SNA forwarding strategies have been considered as candidates for implementation. Experiments in source-route forwarding will probably also be carried out using the multi-protocol capability of this gateway package. As of this writing, the package has been completed and has operated as an IP gateway between the ARPANET and the version 1 ring network.

Another example of a service built on the principle of avoiding on-the-fly protocol transformation is a multi-protocol mail forwarding facility installed this year on the Multics system. Although at the highest level there is agreement among several communities on the format of an electronic message, there are several different protocols for transfer of a message between two hosts: old and new ARPANET protocols, Chaosnet protocols, and some local variants. In some cases, the protocol that is required differs depending on the port over which the message is dispatched. The forwarding service on Multics will accept a message from any source host in any of the various protocols. It will then queue and remail the message to the intended target, using whatever mail-forwarding protocol that host requires. Michael Greenwald did most of this year's work on this facility.

The earlier section on protocol performance mentioned a thesis by Vineet Singh on the design of a service that calculates routes to be placed in a packet at its source. That work has important application for network interconnections that cross administrative lines as well, since an important virtue of a source route system is the ability to control exactly the path taken by a packet. Such control allows a packet to be directed along a path that has appropriate data rate, reliability, delay, or security, and that meets policy requirements (e.g., the packet shouldn't pass through country X or private company network Y). It also allows trouble isolation (by sending packets that have a route that takes them out to a gateway and back to the originator) from any node, a useful facility to reduce finger-pointing as the prime trouble isolation method in multivendor network paths.

As the scale of interconnection of local networks grows, and links extend from one organization to another, a problem arises of identifying the individual users of the network. Proper identification is needed to send electronic mail, and to authorize use of data or other services; it may also be needed to account correctly for use of services. Traditional authorization and naming systems have operated entirely within a single computer system, and have used techniques satisfactory for a population of from a few dozen up to maybe a thousand or so users. When connected to a network, the name of the host computer usually appears as part of the identification, say, when sending mail to someone. These schemes begin to break down when larger numbers of computers each with a smaller number of locally-assigned users appear, and as the cost of computing declines to the point that almost everyone in an organization becomes a computer user. Problems are especially apparent when user names are expressed only as three initials or some locally-known nickname.

This year we began a project to implement an on-line directory service for the MIT community. The goal of this project is to explore techniques for naming people at the scale required at MIT in the future--a community of 15000 people with turnover of perhaps 2000 faces (and names) each year. Two specific services are being developed by Kimberle Koile, with assistance from Felix S. Hsu. First is an interactive directory assistance service that accepts a partial, potentially ambiguous name of someone thought to be at MIT, and responds with a set of possible correct identifications, along with confirming information such as title or department, and office or term address. When the correct person is found, associated with it will be a standard form of that person's name, for use in sending mail and for authorization and accounting. One of the questions to be explored is the extent to which the standard form can be exactly the person's name as he is commonly known--typically a first name, middle initial, and last name.

The second service, based on the file stored by the first service, is a mail forwarding service that eliminates the need to know on which computer system at MIT a person's electronic mailbox resides. Instead, one directs the mail to the person's standard name (perhaps as discovered once before by use of the directory assistance service and then tucked away in a private list of nickname--standard name pairs.)

Operation of these two services at the scale of the entire MIT campus demands that much of the information base be automatically derived from other files: those managed by the registrar of students and the staff personnel department. The project began by obtaining machine readable copies of directory listings from both those sources; continued cooperation with both those organizations as well as the telecommunications office (which publishes telephone books and operates the telephone-based directory assistance service) is anticipated. Also, because the data held by these services is personal in nature, all plans for this data base are being reviewed by the MIT Privacy Committee for suggestions and to spot possibly troublesome points.

As an experiment in practical problems of interconnection, Fredrich Meier zu Sieker, in his undergraduate thesis, designed a gateway to the commercial Telex service, to be usable via the local networks by any authorized person at MIT This project helps focus attention on user identification (for poorly addressed incoming messages) and accounting and authorization for use. An implementation of the service is underway, by Robert Myhill and Lixia Zhang.

As part of our interest in network interconnection, we have maintained a strong interest in the Internet protocol family being developed by DARPA. During this year David Clark took over the job of technical coordination of the architecture of this protocol family, and he is chairing the Internet Configuration Control Board with the responsibility for decision making this this area.

As part of this DARPA Internet project, he has prepared a number of notes which together comprise an informal implementor's guide to the protocols, with particular attention on how to produce a simple implementation that performs well. These memos will be distributed by DARPA in July.

A final project in the area of network interconnection has already been mentioned under the heading of network technology exploration--the use of cable television as a data communication medium. The interconnection aspect of this project is its policy component. There are issues of allocation of cost, of control of access to a limited resource, of provision of related services, and of separation of control over content and carriage. Communication regulations are generally formulated with two distinct classes of service in mind: common carrier, and broadcast. Policy regulations pertaining to the first focus mostly on tariffs while for the second they focus on program content: fairness, community service, etc. Using a cable for data communication involves elements of both kinds of policy regulation, requiring some innovation to deal with the situation sensibly. In addition, in the United States, cable policy regulation is typically handled at the community level, rather than state or federal, and most communities have inadequate resources and expertise to deal with either kind of policy regulation, let alone their interaction.

To shed some light on this area Deborah Estrin just completed a Master's Thesis on the policy problems of using cable television for data communications. The main contribution of the thesis is to outline the range and depth of the policy problems that are involved. In addition, one chapter offers specific suggestions to community policy makers on how to proceed.

5. PORTABLE PERSONAL TERMINAL

During the year, the group has done a preliminary study on the design of a portable terminal, small enough to be carried in the briefcase, if not the pocket. We were interested in exploring ways in which such a terminal could be made realistically useful, given that the small size must involve a severe restriction in the basic capabilities of the terminal. We assumed that individual applications would have to be reprogrammed so that they knew about and could take advantage of the specific features of the terminal. This kind of specialization has been very important when moving from traditional ASCII terminals to more sophisticated bitmap displays; it was our believe that the same specialization would be important for a terminal with restricted rather than enhanced features.

We had in mind two sorts of terminals for this project. The smaller realization consists of a single line of alphanumeric text and an undersize, but traditionally organized keyboard. This terminal configuration appealed to us because products are now appearing on the market with this approximate hardware configuration. The more sophisticated version of the terminal would have a keyboard and a multi-line display, perhaps packaged so that the display could fold up and sit at a traditional angle to the keyboard. The display technology for such a packaging is beyond our ability to fabricate, but we thought it worth exploring the implications of this degree of sophistication, because the range of applications that could be supported was much greater than with a single line of text.

We had in mind particular applications which seemed suitable for a portable terminal. The first application studied was sending and receiving mail, which clearly benefits for a greater flexibility and freedom in the patterns of accessibility. Other applications which would be suitable for such a terminal include a portable appointment book or a database query facility. These alternative applications were not considered in detail.

The first phase of this project concentrated on the simpler of the two terminals, with a single line of display. The first question was how the limited functionality of this terminal should be organized in order to make it maximally useful. Simple experiments with traditional patterns of text display, in which the text is streamed from right to left as a continuous ribbon, quickly eliminated this pattern of display. Because of refresh limitations in the display, this pattern was limited to a very slow reading rate, which the user quickly found frustrating. In a Bachelor's thesis, Russell Houldin experimented with a number of alternative reading modes, using a simulation of an LCD display which he and David Goldfarb programmed for the bitmap display of the Alto computer. This simulation allowed us to experiment before we had implemented any prototype hardware for the terminal itself, and led to the conclusion that a rather specialized reading mode was the proper display pattern for a single line terminal. The reading mode involves dividing the text into small chunks, no more than ten to fifteen characters long (except when a single word is longer), and displaying these chunks in rapid sequence centered in the display. The reader never moves his eye, and can easily be trained to read five to ten chunks per second. In this manner, large quantities of text can be perused quickly.

In order to support this reading mode, the terminal required two special features. First, it required special keys to control the rate of text to display. The reader must easily be able to slow down, or back up and reread something. This requirement in turn implied that the terminal must have an internal buffer, rather then simply displaying text as it came down the telephone line. In fact, the buffer was additionally required in order to permit a peak reading speed which exceeded the data rate of the telephone line. We therefore developed a proposed buffer scheme, in which the application understood about the management of the buffer, and attempted to keep the buffer full, so that as the reader advanced through the buffer, the desired text was always there.

In order to explore some of these specialized features, and to learn something about the actual operation of LCD displays, Clifford Ludwig, in a Bachelor's thesis, undertook the development of a prototype terminal. His goal was not to produce something which was in any sense properly packaged for portability, but rather to produce something which contained hardware and software of the correct power. The prototype contained a Z80 microprocessor, a suitable amount of memory for buffering, and a special display peripheral which consisted of one line, 36 characters long, of LCD display. The prototype was implemented, and certain of the experiments which had been performed on the Alto simulator were recreated on the actual display.

This initial prototype taught us a number of things. First, LCD displays, although clearly desirable from a point of view of power consumption, are marginal for this purpose, because they cannot alter the display fast enough to suit the peak reading speed of a trained user. Blurring and cloudiness occur, even with very careful tuning of the electrical parameters of the driving circuitry. Therefore, more experimentation with the detailed characteristics of LCD displays are required. Second, in order to drive the display properly, specialized display chips are required, which are now available only in a form too bulky to be properly packaged. A commercial version of this terminal would require specialized LSI. Third, a Z80 is clearly powerful enough to provide the display, keyboard, and telephone line controls needed for the terminal.

A related project in this area explored the idea that speech input and output could be an important part of this terminal's function. The intention was not to rely on speech recognition or synthesis, two very difficult problems, but to use speech storage as a way of passing information through the terminal from one human to another without the necessity of dealing with a restricted display or an undersized keyboard. For example, in the context of mail, the user might employ the keyboard to specify the recipient and subject matter of the message, but might dictate the message itself. Alternatively, the terminal could be used as a dictating machine, with later playback and transcription of the information into ASCII representation of the text. Two projects were undertaken to explore the feasibility of integrating speech into this terminal. First, a study was undertaken of speech digitization and storage techniques, to find out how many bits per second of digital information would be required in order to store intelligible speech. Much previous work has been done in this area, but the specific issue we were concerned with was the fact that speech had been transmitted through a telephone line before being digitized, which changes its spectral characteristics. In particular, efficient storage algorithms such as linear predictive coding are unsuitable. We thus did a preliminary experiment with delta modulation techniques, to determine whether they could produce intelligible speech with a reasonable bit rate. In a Bachelor's thesis, David Teller clearly demonstrated that delta modulation is a suitable technique in this context, but he also demonstrated that successful utilization of that strategy in this context would require the development of specialized hardware for the purpose, as opposed to the converter card which we had purchased for initial trials.

The other experiment related to speech involved the development of a specialized modem to connect the terminal to the host. Since we desired to send both digitized information and analog speech over the phone line, it was necessary that the modem be switched off and on by computer control to permit the telephone line to be used alternatively for both digital and analog transmission. Further, at the terminal end, it was necessary to have some form of microphone and speaker which the human could use for sending and receiving speech. Since a telephone line was being used for the transmission of this information, we explored whether a telephone handset could be plugged into the terminal for this purpose. Carol Novitsky did a preliminary study in this area, with the particular goal of determining whether any of the specialized low-speed modem chips which are now available could be used at a bit rate above 300 bits per second. This involved the careful design of high order analog bandpass filters. The results of this experiment were somewhat encouraging, but it is clear that to achieve high bandwidth over a phone line with a very small space and power available in the terminal represent a substantial analog engineering job. We feel that pushing further in this direction should be delayed until we have a better idea of the actual bandwidth requirements.

The final study in this area was an evaluation of mail on the more sophisticated version of the portable terminal, in which many lines of display might be available, perhaps as many as on a traditional CRT. In this case, the study focused on a rather different area of terminal function. For the single line display, it had been assumed that the terminal would be connected to the host using a telephone line whenever the terminal was in use. Because of the space limitations in the terminal, there was no other useful way to imagine utilizing the terminal. However, if the terminal is somewhat larger, it is possible to imagine enough buffering in the terminal that it could serve as an independent computer, being connected to the host only now and then to refresh its internal storage. This is a particularly appealing pattern for mail, since the terminal could be connected to the computer a few times a day to receive any new messages, and then could be carried off to permit the reading of mail whenever convenient. The problem which arises from this pattern of use is one of management of duplicate copies of information. In particular, when a host transfers a piece of mail to the terminal, it is not reasonable for the host to discard its copy of the mail. The terminal, being portable, could be lost or damaged, and it would be inappropriate if mail were lost under these circumstances. Therefore, a synchronizing strategy is required between the copies of the mail stored in the terminal and the host. to ensure that mail is not lost. In fact, it is very important that the host keep a copy of any mail delivered to the terminal, because, as a result of reading the mail, the user may instruct that the mail be disposed of in various ways, perhaps being saved or forwarded or used as the text of a reply. Such functions are much more convenient if the host has its own copy of the mail. Michael Patton, in a Bachelor's thesis, did a first study of the way in which the host and the terminal would cooperate to provide this kind of functionality.

This particular distributed database is a very interesting one for study in general, because it has a very different usage pattern from those traditionally considered in research on distributed systems. Most researchers assume that the database is connected by communications facilities almost all the time, but occasionally partitioned. In this case the database is almost always partitioned, but very occasionally connected. In general, such a partitioned database could not be made to work well, but in the case of mail, because of its particular characteristics, the partitioning does not seem to be a difficult problem. This is an interesting observation, which suggests that more study of application-dependent distributed systems is appropriate.

6. XEROX GRANT ACTIVITIES

An incidental activity of this group is administration of a University Grant of the Xerox Corporation, consisting of 18 Alto personal computers, a Dover laser printer, a file server, an experimental Ethernet local communications network, and related supporting software and facilities. These facilities are now an integral part of the resources of the laboratory as a whole. Eleven of the Altos are currently used in direct support of research projects in four LCS groups, while six are assigned to different groups throughout the laboratory primarily to provide experience with the cultural effect of having high performance personal computers nearby. (The 18th Alto is used as a maintenance spare and for overflow usage by the other groups.) This report briefly outlines the ways in which the Xerox grant facilities have been used. More extensive project descriptions will be found in the annual reports of the individual groups doing the projects.

The Computer Systems and Communications group has undertaken three separate programming projects on the Alto, all related to its protocol effectiveness research. The first, by David Clark, was to create a Dover spooler service, a program that accepts files in the DARPA TFTP/IP protocol, and relays these files to the Dover laser printer in the Xerox Pup protocol. A second project, also by David Clark, was implementation of a very small and fast user Telnet/TCP package that permits the Alto to be used as a remote terminal via internet gateways. Both these systems are now in production use; the Telnet/TCP was recently used to log into a Digital

PDP-11 computer located in Norway, with several networks and gateways in between. The third project, by Geoffrey Cooper, was an implementation of the ANGEL protocol, an IP-based reliable datagram transfer service that uses the soft layering ideas of Cooper's S.M. thesis.

During the past year,the Computer Systems Structures group has used the Altos in three ways. First, the Swallow distributed data storage system repository was built using Mesa on an Alto. The repository is now operational in a test configuration. The availability of Altos was a crucial factor in being able to pursue this work. Second, the group began to use Altos to provide remote, network-attached bitmap displays for single user VAX-750's, using special, high-speed protocols. Third, they used Altos as high performance network terminals, providing 80X60 character, network-attached access to hosts using TCP on the DARPA Internet.

The Systematic Program Development group decided to abandon its attempt to use an Alto as the front end of its specification editor. This decision was based upon the prohibitively bad performance of a prototype implementation, unhappiness with the Mesa 5 programming environment, and a reluctance to increase an investment in what appears to be (for LCS) a dead-end line of machines. Members of the SPDG did get considerable use out of the text preparation facilities of the Alto. Compatibility with text preparation facilities at Xerox PARC greatly facilitated cooperative work with James Horning.

The Functional Languages and Architectures group is developing an advanced computer architecture and a hardware prototype based on "dataflow" principles. That group has found that the Xerox Alto computer is an asset in carrying out its research and in particular they have used the Alto in the following ways:

1) The Draw program has been invaluable in producing high quality figures for many designs. Members of the group have commented that the flexibility of the Draw system has led to qualitative improvements in their designs and documentation;

2) The group is using a stripped-down M68000 microprocessor as an I/O and control processor for each processor of its prototype dataflow machine. By connecting a M68000 to the Alto via an 8-bit parallel port they have been able to quickly tap onto the Alto's existing file system and Ethernet without a large software/hardware effort. A student is working on microcoding the Alto parallel port software to improve the performance of the link.

The group is also considering using the Alto programs Sil, Analyze, Route, and Build to generate wire lists for the prototype. Route's capability to generate Multiwire lists is very attractive since they are planning to use Multiwire boards in the prototype.

James Frankel, a Ph. D. candidate at Harvard University, has been using the Alto's as a distributed computer system on which to implement a prototype of some software ideas in his dissertation. The dissertation, "The Architecture of Closely-coupled Distributed Computers and their Language Processors," deals with the hardware and software design of a shared memory multiprocessor.

The system implemented on the Alto's, a parallel-executing Pascal compiler, compiles syntactic structures of the source code concurrently. Thus, the prototype begins by compiling a program on a single processor. That machine compiles all declarations and then initiates the compilation of nested procedures on other machines passing to them the symbol table that was generated. The initial processor then compiles and produces code for its code block. The machines on which compilations were spawned perform the same sequence of operations in turn.

The generalization of these ideas is that the parallelism for multiprocessor computer systems should come from the data flow rather than the control flow present in programs. Furthermore, the premise is that "coarse data flow," data flow where the data is larger than a single word (for example, the size of a procedure, statement, or expression, for a compiler), allows conventional processors to run in a multiprocessor configuration, reduces communications and, thus, contention for shared memory, and does this without the drawbacks present in data flow machines (inability to deal directly with data structures, difficulties with recursive and reentrant procedures, etc.).

A number of tools were developed during the work on the project. With the cooperation of Roy Levin and other members of the Xerox PARC Computer Science Lab, Mr. Frankel wrote a package to create "boot" files from Mesa 5.0 programs. This package was distributed to all of the universities in the Xerox University Grant Program. The Diagnostic Memory Test, DMT, program was modified to allow an Alto to be down-loaded over the EtherNet only if there are no disks ready in the Alto. A page level file server was written to act as shared memory over the 3 MHz Experimental EtherNet. A client interface to the page server and a stream interface to the client interface were written. When bound with these programs, any Mesa program that was written to access files from disk will access those files from the page server.

The Pascal compiler itself is written in Pascal and compiles into a byte coded instruction set called Pascal Byte Codes. These byte codes are interpreted by the Pascal Byte Machine, PBM, which is itself written in Mesa.

More detailed information is available in Mr. Frankel's dissertation to be completed in August, 1982.

The Dover laser printer is one of the most popular facilities in the laboratory, receiving wide use from almost every computer system of both the Artificial Intelligence Laboratory and the Laboratory for Computer Science. An indication of its popularity is the rate that it consumes paper: 250,000 sheets/month at present. Almost all technical reports, technical memoranda, and submissions of papers to journals from the two laboratories are prepared with the help of this printer.

Publications

1. Clark, D., "Local Networks," to be published in Computer, accepted April 1982.

2. Corbató, F., "An MIT Campus Computer Network," Campus Computer Network Group Memo No. 1, MIT, Cambridge, MA, 1981.

3. Corbató, F., "Time Sharing," Encyclopedia of Computer Science, A. Ralson, Editor, Second Edition, van Nostrand Reinhold Co., New York, in press.

4. Estrin, D., "Data Communications via Cable Television Networks: Technical and Policy Considerations," MIT/LCS/TR-273, MIT Laboratory for Computer Science, Cambridge, MA, May 1982.

5. Saltzer, J., "Communication Ring Initialization Without Central Control," MIT/LCS/TM-202, MIT Laboratory for Computer Science, Cambridge, MA, December 1981.

6. Saltzer, F., "On the Naming and Binding of Network Destinations," International Symposium on Local Computer Networks, Florence, Italy, April 1982, pp. 311-317.

7. Saltzer, J., Clark, D., and Pogran, K., "Why a Ring?" Seventh Data Communications Symposium, Mexico City, Mexico, October 1981, pp. 211-217.

8. Singh, V., "The Design of a Routing Service for Campus-Wide Internet Transport," MIT/LCS/TR-270, MIT Laboratory for Computer Science, Cambridge, MA, August 1981.

9. Wright, K., "A File Transfer Program for a Personal Computer," MIT/LCS/TM-217, MIT Laboratory for Computer Science, Cambridge, MA, April 1982.

Theses Completed

1. Baldwin, R., "An Evaluation of the Recursive Machine Architecture," M.S. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1981. (Also S.B.)

2. Estrin, D., "Data Communications via Cable Television Networks:

Technical and Policy Considerations," M.S. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

3. Houldin, R., "Formats and Controls for a One-Line Computer Terminal Display," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, August, 1981.

4. Ludwig, C., "A Personal Portable Terminal," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, February 1982.

5. Martinez, D., "A Central Switcher for Message-Oriented Computer Input/Output," M.S. thesis, MIT Department of Electrical Engineering and Computer Science, May 1982. (Also S.B.)

6. Meier zu Sieker, F., "A Telex Gateway for the Internet," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

7. Patton, M., "Integrating Disconnected Personal Computers into an Electronic Mail System," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

8. Powell, R., "Microprocessor-Based Floppy Disk Controller," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

9. Singh, V., "The Design of a Routing Service for Campus-Wide internet Transport," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, August 1981.

10. Teller, D., "Efficient Storage of Digitized Speech," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

11. Tou, F., "Information Retrieval in a KL-ONE Data Base," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1982. (Also S.B.)

12. Wright, K., "A File Transfer Program for a Personal Computer," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, April 1982.

13. York, W., "Command Completion in the Multics Environment," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1982.

Theses in Progress

1. Cooper, G., "An Argument for Soft Layering of Protocols," M.S. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1982.

2. Greenwald, M., "Operating System Support for Closely Cooperating Talks," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1982.

3. Koile, K., "An On-line MIT Directory Service," M.S. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected June 1983.

4. Konopelski, L., "Implementing Internet Remote Login on a Personal Computer," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected August 1982.

5. Lopez, L., "Gateway Congestion Control," M.S. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected January 1983.

6. Roush, P., "Computerized Scheduling of Intramural Sports," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected August 1982.

Talks

1. Chiappa, N., "Developments in Fault Tolerant Local Networks at MIT," South Padras Island, Texas, February 1982.

2. Clark, D., "The LCS Ringnet Project,"

   University of Strathclyde, Glasgow, Scotland, April 1982;

   University of Kent, Canterbury, England, April 1982;

   Siemens, Munich, Germany, May 1982.

3. Clark , D., "Problems in Local Network Interconnection," Los Angeles, CA., November 1981.

4. Clark, D., "The ARPA Internet Project,"

   Computer Laboratory, Cambridge University, Cambridge, England, May 1982;

   IBM Zurich Research Laboratory, Zurich, Switzerland, May 1982.

5. Corbato, F., Participation in the Communications Policy Seminar on Ad Hoc Networks of Microcomputers, MIT, Cambridge, MA, April 15, 1982

6. Martin, E., "Network Handling on a Small Computer Running UNIX," Internet Working Meeting, Cambridge, MA, March 25, 1982.

7. Saltzer, J., Session Chairman, "Local Networking Approaches," ACM 81, Los Angeles, CA., November 1981.

8. Saltzer, J., "Technology, Bureaucracy Avoidance, and Distributed Computer Systems": "Intriguing Ideas About Computer Systems"; and "Heretical Ideas About Local Networks," 1982 George Forsythe Lecturer, Stanford University, Stanford, CA., January 1982.

9. Saltzer, J., "Technology, Bureaucracy Avoidance, and Distributed Computer Systems," invited lecture, First Colombian conference on computer science, informatics and related sciences, Bogota, Colombia, South America, March 1982.

Committees

1. Clark, D. D., DARPA/TCP Working Group (Chairman)

2. Chiappa, J. N., DARPA/TCP Working Group

3. Martin, E. A., DARPA/TCP Working Group

4. Saltzer, J. H., DoD/DDRE Security Working Group Member

5. Saltzer, J. H., Chairman, 9th ACM Symposium on Operating Systems Principles

6. Saltzer, J. H., Program Committee, IFIP/TC6 Working Conference on Interconnected High Performance Personal Computing Systems

COMPUTER SYSTEMS STRUCTURES

Academic Staff

D.P. Reed, Group Leader

Research Staff

M. Greenwald

Graduate Students

B. Coan	J. Stamos
D. Daniels	D. Theriault
W. Gramlich	C. Topolcic
K. Sollins	

Undergraduate Students

R. Kukura	D. Solo
M. Novick	K. Yelick
C. Rubin	

Support Staff

D. Fagin

Visitors

O. Hvinden	L. Svobodova

1. INTRODUCTION

During the past year, the effort of the Computer Systems Structures Group has been focused on development of tools and substrates appropriate for development of distributed applications systems. We find that the most interesting research problems arise from trying to exploit two "insurmountable opportunities": 1) the opportunity to connect two autonomously managed independent computer systems with a computer network in order to share data, and 2) the opportunity to use high performance networks and specialized server computers to build multi-user computer systems that are modularized in a way that has not been possible before with single, large mainframe computer systems.

The first opportunity, data sharing, became apparent when the ARPANET computers first began to exchange data. The level of interconnection on the ARPANET however, has never been particularly high. The best example of distributed applications within the Arpanet has been the mail system. Higher level applications that share databases across the network have been rare and extremely ad hoc. Our goal in this area is to develop substrates, such as the Swallow prototype described below, that can support applications at independent sites that can be later combined into larger applications while maintaining the autonomy of the original applications. The new research problems in this area result from two characteristic issues: the first issue is autonomy, that is, the fact that there is no "central administrator" who controls what is done on each computer in the distributed system, while the second issue is "growth by federation", that is, that adding a single gateway between two independent networks of autonomous processors may all of the sudden create a single system with complete interconnection. Although this federation process is easy at the hardware level, the software structures developed for distributed systems have been hierarchical, with naming, protection, concurrency control, failure recovery, etc., managed by what amounts to a single central authority. Trying to combine two hierarchies results in a hierarchy that no longer functions, because there is a new ambiguity -- "who's on top?".

The second opportunity modularization arises from new local network technologies, workstation technologies, etc., that allow the construction of what might be called "server-oriented systems". For reasons of reliability, economy of scale, and flexibility, it is often convenient to design a distributed system consisting of a set of workstations with no secondary storage or only very small amounts of local secondary storage, with the bulk of secondary storage being provided by one or more shared, specialized data storage service machines. Similarly, specialized services such as printers, image scanners, or wire-wrap machines, may be attached to the net rather than directly to any particular work station. These new structures present problems of reliability, performance, protection, and coordination that differ significantly from the same problems as they appear in centralized time-sharing systems with many attached peripherals and file storage devices. The Swallow repository, which is a specialized data storage server computer, is a prototype of one kind of shared service.

In addition to developing substrates and servers, such as those above, we have also been working on several specialized network protocols. These protocols called non-fifo protocols, achieve extremely high performance and extreme simplicity by ignoring the conventional wisdom of protocol design. Instead of many layers of protocol implementing virtual circuits, these protocols use end-to-end datagram transport, and involve the application in error recovery, flow control, and coping with out of order packet arrival. Our initial experience with these protocols leads us to believe that such protocols will be necessary to exploit the potential of high bandwidth local networks, long delay high bandwidth satellite connections, and internetwork coupling.

In the following sections, we summarize the results of the past year's work on Swallow, protection and authentication in distributed systems, protocol design, naming in distributed systems, and debugging of distributed systems.

2. THE SWALLOW SYSTEM PROTOTYPE

Over the past year, work on the Swallow system has focused on the development of the Swallow system repository prototype. The Swallow repository is a specialized data storage server that provides stable storage to any number of clients on a network. A Swallow system may contain any number of repositories and each user may use any subset of the available repositories to store his data. The Swallow repository participates in concurrency control and recovery algorithms designed by Reed [1] [2] to provide multi-site atomic actions. The Swallow repository design was also conceived with the intention that it could be based on write-once storage media such as optical disk technology now being developed in a number of places.

The Swallow repository was built on an Alto with a special additional large disk drive since that hardware was available to us at the time. Although the rest of the Swallow system was not available to use the repository, we began to test and tune the system during the past spring semester, so that it could be incorporated easily into the Swallow system once the rest of it is constructed.

The design of the Swallow repository; particularly its use of write-once disk, required the development of a new storage organization (called append-only storage) which naturally supports the object of Swallow which have dynamically varying size and which have multiple versions over time. This concept, first introduced by Reed in his doctoral thesis [1] and developed by Reed and Svobodova, is documented in a paper recently published by Svobodova [3].

The client interface to the Swallow system is provided by a software module in each computer called the broker. The design of the broker was begun during the past year, but it was decided that the final design decisions would have to wait for the arrival of appropriate workstation computer hardware. At this point it seems likely that such workstations will be implemented on VAX II/750's which will arrive during the coming summer. Our next task then will be to finalize this design and determine how to integrate it with the operating system (UNIX) of the VAX.

3. PROTECTION AND AUTHENTICATION

During the past year we completed and tested our authentication server prototype and began to see how it could be used in securing various communications that currently go on in the lab. One result of this was a bachelor's thesis by Solo, who investigated the problem of securing a file transfer protocol [4].

The approach of using authentication servers has a flaw, which we view as a very important one. This flaw is that the authentication server used to authenticate one party to another is in a sense a central authority. As distributed systems grow larger and cross organizational and governmental boundaries, there may be no real single trusted central authority. Problems of authentication and protection across such boundaries will require and different and novel solutions. During the past year Topolcic has developed a technique for creating "digital guarantees" that can be used where a client and a server need a mechanism to enforce the satisfaction of remote requests in a decentralized system without such a centralized authenticator.

Consider this scenario. In a network of autonomous computers having varied resources, a client may request a service from some other node, the server. Since autonomous nodes within different organizations may be mutually suspicious, and since there may exist no universally trusted authority, some decentralized mechanism is necessary to assure the client that its requests will be honored. Topolcic examines some failures that can interfere with fulfillment of the remote requests and methods to control them.

One source of failure is the dishonesty of the server, which might return incorrect results, or may ignore some commitment it had previously made to the client. Cryptographic "Digital Signatures", as proposed by Needham and Schroeder [5], attempt to provide a binding "guarantee" from a server to a user. Needham and Schroeder's approach requires that the encryption keys be protected for extended periods of time. Such long-term protection, regardless of the security of the encryption technique, demands administrative controls that are difficult to identify and impossible to prove correct. Topolcic proposes a system of guarantees based on a hard to duplicate yet completely public characteristic function (rather than encryption) and a distributed method of monitoring and punishment based on a

"User's Group" rather than a universally accepted judge. The characteristic function of a guarantee is placed by the server into the next guarantee it issues, forming a linked list which cannot be modified without changing the latest one issued, whose uniqueness is verified with real-time authentication. The User's Group is a collection of clients which monitor the server, exchange information about it, and enforce punishment by boycotting it if a member proves the server's dishonesty. A minority of non-participating or dishonest members cannot affect the correctness of the actions of the majority.

4. NAMING WITHOUT HIERARCHY OR A CENTRAL AUTHORITY

In nearly every computer system the naming mechanism consists of a hierarchy with pieces of a single global name space assigned to each user who can then assign names within those parts to objects of his own interest. A global name space presents problems in a system that grows by adding communications points between preexisting but independent distributed systems on independent networks. Where each system had its own global name space in which all names were unique, the combined system has name conflicts. Where each system had a central authority that partitioned the name space before, there are now two or more central authorities in the combined system. Thus the notion of a hierarchy tends to break down in these <u>federated</u> <u>systems</u> (which result from interenterprise linkage, in particular). A new approach to naming is needed.

If we look at the "human distributed system", we find a potential for similar problems in human language. Here, a distributed system is analogous to a human community, and the computers are analogous to individual people. As thousands of years of human history have shown, people have little trouble with the problems with integrating name spaces that so confound computer systems. We felt that by exploring this analogy, a new approach to computer naming of things could be developed, which would be flexible, natural, and free of the problems of hierarchy.

Sollins has been exploring these ideas in her Ph.D. thesis which was begun during the past year. Although the ideas are still at an early stage, they promise to be significant.

Sollins proposes contexts as the system provided tool for name management. In addition to the goal of non-hierarchical naming mentioned above, Sollins' work will provide a unified naming framework for all entities in a distributed computing environment where each node must be capable of independent operation without dependence on others. This independent or autonomous operation leads to the conclusion that names cannot be guaranteed to be unique. It is this assumption of autonomy that has led away from more traditional remote name servers. All these forces combined have led to the model of contexts proposed in this thesis. There are two sorts of functions provided by contexts. First, a context might translate a name into something else, either another name or an address. Second, a context might answer the question of whether two names name the same entity. Contexts allow names to be assigned to any entities, people, processes, data, or whatever else needs to be named.

There are several different kinds of names that are used commonly in naming during human interactions. These are modeled in the naming framework provided. One is the ability of the name user to assign nicknames. Another is the ability to name by description. In this case, an entity might be described by a collection of descriptive attributes. The namer will name the entity by indicating a logical combination of these descriptions. A third form of name is what in this work is called generic naming. This is a means of naming a class of entities by using a single name. An example of such a class is the set of implementations that provide a particular service, although the entities named by a generic name need to be the same type of entity. Each namer should be able to use whatever name he chooses for the entities he wishes to name. In fact, although superficially these three kinds of names appear to be different in nature, they are not. Any two can be described in terms of the third, or all in terms of generalized names or labels. For instance, assuming there are only generic names, a nickname is simply a generic name that names only one entity, which also has at least one other name. The entity named by a description is simply the intersection of those entities named by a collection of generic names, one for each attribute in the description.

There are a number of issues related to contexts and naming that remain to be investigated. The following is a partial list:

1) the relationship between naming as provided by contexts and protection and authentication.

2) how contexts will be used, individually and in combination with each other.

3) how and when contexts should and should not be shared and by whom or what.

4) a detailed example of the use of contexts.

This work is in progress and will continue on the assumption that contexts are a useful mechanism for name management in future systems.

5. DISTRIBUTED DEBUGGING

Gramlich has undertaken a Ph.D. thesis to investigate a debugging methodology called *checkpoint debugging*. Basically, checkpoint debugging works by taking regular checkpoints of a program. A checkpoint consists of a fixed part and incremental part. The fixed part of a checkpoint consists of a single consistent snapshot of the relevant program state. The incremental part of a checkpoint consists of a sequential recording of all program input since the time of the program snap-shot. When a program failure occurs, it is possible to use the checkpoint information to repeat deterministically the failure as many times as necessary to locate the program failure. This is done by going to a previous checkpoint, loading the fixed part, and reexecuting the program using the incremental part for program input. The major advantage of checkpoint debugging is that it converts a large class of non-deterministic failures (i.e., non-repeatable) into deterministic failures (i.e. repeatable). It turns out that it is much easier for a user to locate and correct a deterministic failure than a non-deterministic failure. A trial implementation of this debugging system will be implemented in CLU for Berkeley Unix.

6. NON-FIFO PROTOCOLS

The so-called end-to-end argument [6] is a protocol design rule that says, in effect, that the application usually knows best how to cope with such problems as loss of messages, protection, flow control, duplicate message detection, coping with messages arriving out of order, and so forth, which traditionally have been in the domain of the communications subsystem. A corollary to the argument is that the communications subsystem may be paying a very high performance price that results from implementing solutions to these problems at too low a level in the system. In fact, in order to have any layering of function at all, it is necessary to place some functions at least inside the communications system and below the application. But if the application knows best, and implements all these functions for itself, there is little or nothing left in the communications subsystem to layer.

This line of reasoning seems to be borne out by a couple of protocols we developed recently for two specialized applications. By exploiting natural properties of the applications themselves, we were able to accomplish the flow control and error control functions involved in communication in a much simpler and more efficient way. Similar simplifications and efficiencies were obtained in the protocols developed for the Swallow distributed data storage systems communications needs [7].

The first of these specialized protocols was a protocol called BLAST. We observed that most file transfer protocols, even when implemented on very high bandwidth local networks, such as a ten megabit-per-second ring network, had disappointingly slow information transfer rates. For example, a file transfer from one Alto to another on a three-megabit-per-second Ethernet rarely exceeds 75,000 bits per second, while the underlying communication medium and disks are capable of much higher rates. The "accepted wisdom" for implementing file transfer protocols is to access the file as a sequential stream of characters, transmit each character successively over a virtual circuit between the two computers, and store the file sequentially on the remote machine using a stream-oriented file system interface. The stream interface to files and the virtual circuit, of course, are implemented by fairly complex mechanisms that take the raw blocks of the disk or the raw packets of the network and transform them into something that is quite different, a reliable ordered stream of bits. This transformation, or extraction, is not natural. The result is that the application has very little control of the timing of what is going on and the timing itself is critical. To cope with a lost packet in the network, for example, the network virtual circuit implementation introduces a small amount of delay in the communications. The result of such delay will be delay in accessing the next byte of the file, but delay in accessing the file can result in a significant real time delay while the disk rotates one whole revolution. This, in turn, disrupts the smooth flow of data to the receiver across the network, which can effect both the flow control to the receiver, and also the rate at which packets can be stored on disk at the receiver.

Our new BLAST protocol is based on a very simple idea. The sender transmits the blocks of the file each in a separate packet labeled with a block number of the file. Since the block number is in each packet, the receiver can place each packet directly in the file at the time he receives it. If packets are lost in the network, there is no need for the sender to retransmit those packets right away -- instead the sender can continue to transmit the rest of the packets of the file. When the sender thinks that all the packets have been transmitted to the receiver, he polls the receiver with a single packet and the receiver responds with a packet that indicates the set of file blocks that remain to be transmitted. The sender then rereads just those blocks of the file that need to be retransmitted and resends them. This process converges after a few rounds. Neither end needs buffering for error control or reordering. Duplicate packets are not a problem since they may be stored again in the same place and reordered packets just get stored in a different order into the file. The network and communication system serve as a way of getting packets from one end to the other only.

To understand why this protocol is interesting consider a satellite link. Typical satellites have channel capacities of maybe up to 50 megabits per second, but the delay due to speed of light is on the order of seconds from end to end. Satellite channels also tend to have a fairly high probability of packet loss. Traditional stream protocols do not cope well with this combination of high bandwidth and long delay. A single packet loss discovered at the receiver may require one or two seconds before it can be filled in by a retransmitted packet. Meanwhile tens of millions of bits have been transmitted over the network and must be buffered at both the receiver and sender until the retransmitted packet arrives (thus megabyte buffers are needed). In order to maintain throughput on the order of 50 megabits per second, these millions of bits must then be written instantaneously onto the disk at the receiver. In contrast the BLAST operates quite reasonably on such a network with no buffering at the receiver or sender at all. The round trip delay only affects the final polling to determine if all packets have arrived, and for long files, this is negligible. The instantaneous dumping of the entire receive buffer to disk will no longer be necessary.

A similar protocol has been developed so that a remote single user computer can access a bitmap display such as that on the Alto across a high performance local network. In BLINK, each end maintains a copy of the bitmap for the screen. As the computer changes regions of its bitmap, the updated regions are transmitted to the remote display. Since updates to non-overlapping regions may be applied to the display bitmap in either order, a lost packet need not delay processing of later packets arriving at the receiver. Periodically, every hundred milliseconds or so, the display sends a packet containing a version number for every region on the display. The computer then retransmits any portions of the bitmap that have not made it to the display bitmap. The performance arguments for this approach are similar to those for BLAST.

References

1. Reed, D.P. "Naming and Synchronization in a Decentralized Computer System," MIT/LCS/TR-205, MIT Laboratory for Computer Science, Cambridge, MA, September 1978.

2. Reed, D.P. "Implementing Atomic Actions on Decentralized Data," Journal of the ACM, to appear 1982.

3. Svobodova, L. "A Reliable Object-Oriented Repository for a Distributed Computer System," ACM Eighth Symposium on Operating Systems Principles, Pacific Grove, CA, 47-58, December 1982.

4. Solo, A. "User Authentication and Security Modifications for TFTP," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

5. Needham, R., and Schroeder, M. "Using Encryption for Authentication in Large Networks of Computer," Communications of the ACM, 21, 12, December, 1978.

6. Saltzer, J.H., Reed, D.P., and Clark, D.D. "End-to-end Arguments in System Design," Proceedings of the Second International Conference on Distributed Computing Systems, Paris, France, April 1981.

7. Reed, D.P. "SWALLOW: A Distributed Data Storage System for a Local Network," in Local Networks for Computer Communications, North-Holland, New York, NY, West, A and Janson, P. (eds.), 335-373, 1981.

Publications

1. Reed, D.P. "Implementing Atomic Actions on Decentralized Data," accepted for publication by Communications of the ACM, New York, NY, 1982.

2. Reed, D.P. and Svobodova, L., "SWALLOW: A Distributed Data Storage System for a Local Network," in Local Networks for Computer Communications," A. West and P. Janson (Editors), North-Holland Publishing Company, New York, NY 1981, pp. 335-373.

3. Saltzer, J.H., Reed, D.P., and Clark, D.D., "Source Routing for Campus-Wide Internet Transport," in Local Networks for Computer Communications," A. West and P. Janson (Editors), North-Holland Publishing Company, New York, NY, 1981, pp. 1-23.

4. Schiffenbauer, R., "Debugging in a Distributed System," MIT/LCS/TR-264, MIT Laboratory for Computer Science, Cambridge, MA, September 1981.

5. Svobodova, L., "A Reliable Object-Oriented Repository for a Distributed Computer System," ACM Eighth Symposium on Operating Systems Principles, Pacific Grove, CA, December 1982.

Theses Completed

1. Daniels, D., "Query Compilation in a Distributed Database System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, February, 1982 (also S.B. degree).

2. Lederman, A., "A Pascal Structure Oriented System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1981.

3. Schiffenbauer, R., "Debugging in a Distributed System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1981.

4. Solo, D., "User Authentication and Security Modifications for TFTP," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

5. Stamos, J., "Grouping Strategies for an Object Oriented Virtual Memory," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, February, 1982 (also S.B. degree).

6. Ulloa, M., "A Window Manager for Microcomputers," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

7. Weiss, S., "Managing Software Evolution in an Object-Oriented Environment," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

Theses in Progress

1. Gramlich, W., "Checkpoint Debugging," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected date of completion, June 1983.

2. Ketelboeter, V., "Forward Recovery in Distributed Systems," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected date of completion, August 1982.

3. Mendelsohn, A., "A Framework for User Interfaces to Distributed Systems," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected date of completion, June 1983.

4. Sollins, K., "Name Management in a Distributed System," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected date of completion, June 1983.

5. Topolcic, C., "Ensuring the Satisfaction of Requests to Remote Servers in Distributed Computer Systems," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected date of completion, August 1982.

Conference Participation

1. Reed, D.P., Program Chairperson, ACM Eighth Symposium on Operating Systems Principles, December 1981.

2. Svobodova, L., "A Reliable Object-Oriented Repository for a Distributed Computer System," ACM Eighth Symposium on Operating Systems Principles," Pacific Grove, CA, December 1981.

Talks

1. Sollins, K., "Distributed Computing at MIT ", University of Southern California, Los Angeles, CA, December 1981.

2. Reed, D.P., "Protection Issues in Distributed Systems," talk to JIPDEC group on security, audit and control, Cambridge, Massachusetts, April 1982.

3. Reed, D.P., "An Overview of Distributed Systems Research at MIT LCS," Siemens, Munich, Germany, May 1982.

4. Reed, D.P., "Non-FIFO Protocols or Streams Considered Harmful," IBM Zurich Research Laboratory, Zurich, Switzerland, May 1982.

Committee Membership

1. Reed, D.P., Program Chairperson, ACM Eighth Symposium on Operating Systems Principles, December 1981.

EDUCATIONAL COMPUTING GROUP

Academic Staff

H. Abelson, Group Leader	A. diSessa

Research Staff

D. Neves

Visiting Staff

M. Schneider

Undergraduate Students

L. Bagnall	M. Hailperin
J. Dempsey	G. Kiczales
R. Hyre	E. Tenenbaum

Support Staff

D. Tatar

1. INTRODUCTION

1981 was the year in which the world discovered educational computing. Suddenly, it is taken for granted that students will have easy access to personal computers, and that the ability to program a computer will soon be regarded as a "basic skill." For those of us who have long propounded such ideas to a skeptical educational establishment, it is almost shocking to see our most futuristic visions easily asserted as the obvious course of events by the pundits of *Time* and *Newsweek*. 1981 was also the year that the Logo computer language, developed at MIT over the past 12 years, became widely available as a commercial product, and is being hailed as a major breakthrough in computers and education. Many of the Group's members have been involved in Logo development, and it is gratifying to see the speed and enthusiasm with which Logo is gaining popularity. Abelson's *Logo for the Apple II*, published in May, has provoked wide interest, and a number of additional books about Logo and Logo programming are already underway.

But despite this optimism and enthusiasm, there is need for serious concern. For, as the world has discovered educational computing, it will also soon discover that the mere proliferation of computers (even Logo computers) is no magical cure-all for educational problems. In the LCS Educational Computing Group, we are confronting major issues that will need to be addressed before computation can play an effective role in education:

1.1. Computer-based Education Requires a New Perspective

We have long argued that a computer-based approach cannot be simply a matter of transposing traditional material to a new medium. The computational framework is at once a challenge and an opportunity

- *to change the nature of the knowledge being transmitted:* in science and mathematics, especially, to take advantage of constructive, process-oriented formulations of technical ideas, which are often more assimilable and more in tune with intuitive modes of thought than the axiomatic-deductive formalisms in which these ideas are usually couched

- *to change the relation between the student and the knowledge:* to structure richly interactive environments in which "learning through discovery" becomes more than just a well-intentioned phrase, which allow for the personal involvement and agency that we are convinced is essential to truly effective education

This past year saw the publication and enthusiastic response to Abelson and diSessa's book *Turtle Geometry*, a computational approach to mathematics ranging from elementary geometry through General Relativity that has been acclaimed as "the first step of a revolutionary change in the entire teaching/learning process." We think it is crucial to continue to explore the way in which computation can provide an intellectual framework for substantial reformulations of traditional disciplines. During the fall semester, we led a seminar for MIT undergraduates, aimed at developing similar new treatments of topics in physics. Our long-range plan is to incorporate the computational perspective in a substantial way into the MIT undergraduate curriculum, beginning with experimental alternatives to the freshman calculus and physics courses. Much of our work in system development (discussed below) is aimed at providing suitable computational tools to support such an experiment.

1.2. Teacher Training and Curriculum Development Remain Serious Problems

Very little of the vast amount of computer-aided instruction that is currently making its way into schools reflects the kind of intellectual reformulation envisioned in the paragraphs above. But valuable as a book such as *Turtle Geometry* is as a source of ideas, it is a great leap, even for the most gifted and diligent teacher to transform that into a curriculum. And "model" mathematics and physics courses at MIT are of extremely limited value for elementary and secondary school teachers. More formidable still, is the challenge of realizing the educational potential of the computer in the home. In addressing these issues, we look forward to working with Dr. Sylvia Weir, who will be joining our group at LCS during the coming months. Dr. Weir and her research team have been active in teacher training and curriculum development (as well as in computer-based instruction for special needs students) and we expect that her efforts will enable the group to make major contributions in these areas.

1.3. Educational Computer Systems Are Not Sufficiently Powerful or Flexible

It may seem strange that we began working on a new computer system even before Logo became commercially available. But Logo, a language tailored for the personal computers of 1981, is not sufficient for the next round of computer applications. The increasing use of personal computers in homes, schools, and workplaces ensures that we will very soon reach the point where most of the people who interact directly with computers are non-computer specialists, people for whom the computer must be a useful tool without demanding inordinate computational sophistication or effort. These people will require computing systems with broad functionality, including, for example,

- the ability to edit and manipulate text, and to store and retrieve text using a structured filing system

- the ability to search and manipulate data bases

- the ability to use and to modify pre-written programs

- the ability to write new programs, thereby extending the system's underlying capabilities

The traditional approach to designing such tools has been to adhere to the paradigm of the general-purpose time-sharing systems of the 1970's, that is, to produce systems whose functionality derives from the fact that they incorporate a large number of special-purpose independent subsystems. We are convinced, however, that most non-specialist users of computers are best served by providing a computational environment that is *integrated* and *coherent*, in which all of the basic capabilities can be assimilated to a single, uniform, easily understood computational scheme.

Over the past year, under DARPA funding, we have been working on a system called Boxer, which is designed to be the base for an integrated computational environment that provides a broad array of functionality for non-expert users. The Boxer language, like Logo, sits in the line of Lisp inspired languages, but with some crucial distinctions. In particular, Boxer makes the user interface much more integral to the meaning of the system than any previous language. This allows the user's stance toward the system to be much more the naive realism of "what you see is what you have," and thus enhances communication of the model of the system that we intend the user to have. Boxer also makes use of a pervasive spatial metaphor in which linguistic structures and relations are mirrored in the spatial relations shown on the screen.

Although we are most interested in Boxer's potential for educational use, it should be clear that such a system will have a applications over the whole range of personal computation. In the face of the advantage of an integrated system, in which learning any one functionality automatically carries competence into other areas, it is striking that any examples of integrated computational environments suitable for beginning users do not exist. Instead, most current work on designing "usable systems" still aims largely at providing separate systems for separate functions, apparently because these different functions have been associated with different pre-computer technologies and because development projects tend to regard their task as one of providing only the functionality that has traditionally found a place in some individual "job category."

Here is a summary of our progress to date on the integrated system:

- We have outlined the semantics for Boxer and implemented an interpreter for it on the Lisp Machine.

- During the summer and fall of 1981 we designed, implemented, and tested (on naive subjects) a prototype editor system on the Lisp Machine.

- During the Spring of 1982 we began implementation of a second prototype, including an editor, parser, and data base functions.

- We are beginning to write more substantial programs in the system which will give us needed feedback on facilities needed and potential problems with the user interface.

- We have experimented with on-line documentation, taking advantage of the unique capabilities of the system.

- We have implemented (in Lisp, on the Lisp Machine) examples of advanced educational physics systems, which we intend that Boxer will eventually be able to support.

In the following sections, we begin by setting forth some general principles which have guided us in the design of a system for non-expert users. We then sketch the Boxer system as it currently stands. Finally, we outline the next steps in the implementation effort as we see it and our projected activity over the next year.

2. DESIGNING SYSTEMS FOR NON-EXPERT USERS

What issues face one in the design of an integrated computational environment for non-expert users? One stands out above all others: that the user should perceive the system as understandability and simple. While efficiency and power can serve as measures for systems intended for experts, learning and understanding are paramount in a system that can be used by the majority of people. In order to design effective computational environments we must therefore try to understand the mental models that people form of these and other complex systems.

Unfortunately, cognitive science and psychology have not yet provided the wealth of well-elaborated theory and empirical studies of understandability one would like to have before beginning a design exercise. Although there are the beginnings of such theories [6], we are still at a stage of announcing principles at best just before applying them, and in fact, often explaining those principles through their application. Our research on the Boxer system is as much an attempt to explore the principles of designing computer systems so as to be understandable as it is an attempt to develop a particular system. Indeed, from the larger perspective of "engineering for understandability," the possibility of designing computational systems may well motivate theories of understandability in the same way that the technology for fabricating steam engines prompted the development of thermodynamics.

In his paper [1], A. diSessa sets forth principles of understandability for integrated computational environments, together with their application to the design of Boxer, identifying paradigmatic classes of models that users make of complex systems, each with its own strengths and weaknesses. His analysis shows how the crucial notion of "simplicity" takes on a much more textured and complex character than might be the case if one had used less developed notions of what it is to "understand." It also suggests the possibility (even necessity) of using different models of "understanding" for different purposes, and for a gradual shift in the kind of model employed as a user becomes more experienced.

An expert's view of "simplicity" is likely to be linked to issues of consistency, and to the ease of maintaining a simple computational model that provides a complete image of the system's behavior. But no matter how simple such a model might be *from a computational point of view*, it is bound to be complex to the beginner, who likely has no intuition for the kinds of things the model might explain. Rather than attempting to start with a picture of the whole, the novice user needs to be able to isolate manageable portions of the system via a set of working hypotheses and master these sections before moving onwards. We believe that this is possible in a system where the user can initially invoke a set of familiar real-world models for the machine and then modify these to accommodate to the less familiar aspects of computing. A major goal of "engineering for understandability," is thus to smooth the way from simplicity as it appears to the beginner (that is, in the familiar) to simplicity as it appears to the expert (that is, in the logical and consistent).

3. A BOXER OVERVIEW

This section highlights some of the important features of the Boxer system as it is currently being implemented on the Lisp Machine.

3.1. Boxer Uses a Consistent Spatial Metaphor

Particularly since the advent of bitmap displays, the visual medium has served a more and more important role at the interface between man and machine. But surprisingly little use has been made of the medium to develop and support user models, rather than simply to expand the bandwidth of the user interface. In contrast to pop-up menus and iconic mnemonics, we use the video screen to attack the fundamental problem of understandability of the basic organization and operation of the computational environment.

The Boxer system is structured in terms of a comprehensive spatial metaphor. In particular, spatial organization has strong semantic content; elements of the environment are *places*, and their spatially visible relationships have structural meaning in the environment. Perhaps most important, all computational objects are represented, created and manipulated in essentially the same way, and the user can for almost all purposes pretend that the objects *are* their visual representation. What we want the user to see on the screen is, as close as we can arrange it, the computational system itself rather than a multiply-filtered or side-effect dominated view of it. Taking this "naive realism" so seriously, in fact, separates this endeavor most strongly from all previous computer language designs. Our goal is to provide a general-purpose and powerful computational tool that the beginning user will find at once attractive, that is, possible to understand and control immediately, and instructive, that is, whose less familiar aspects can be learned incrementally and naturally.

3.2. Data is Organized Spatially

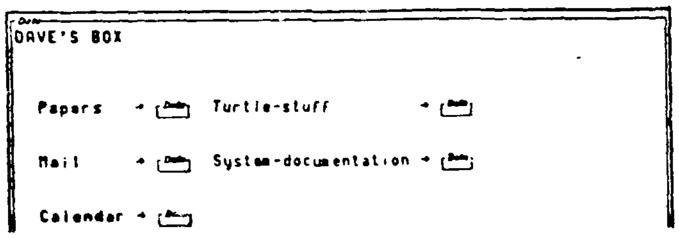


Figure 1

Figure 1 shows the Educational Computing group box. Like all boxes, it contains text (i.e. characters) and other boxes. The shading on the small boxes indicates that these boxes are *shrunken*, i.e., that we do not see there details. The arrows denote names for boxes. (The arrow is the general way that assignment is indicated in Boxer.) By using the Lisp Machine locator device (the mouse) the user can enter any of these inferior boxes, or exit the current box to get to the superior box. This box shows that sharing among people is accomplished by having everyone operate in a single box (virtual world).

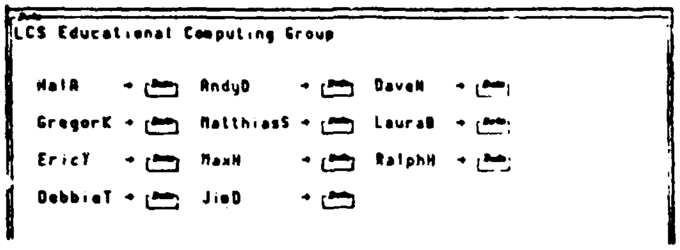


Figure 2

We can enter into Dave's box (Figure 2) by positioning the cursor over the box and hitting a key. At this level we can see that Boxer acts as a directory in normal operating systems such as Unix. One important difference is that the Boxer structure *is* the directory, and the user moves around *within* the structure.

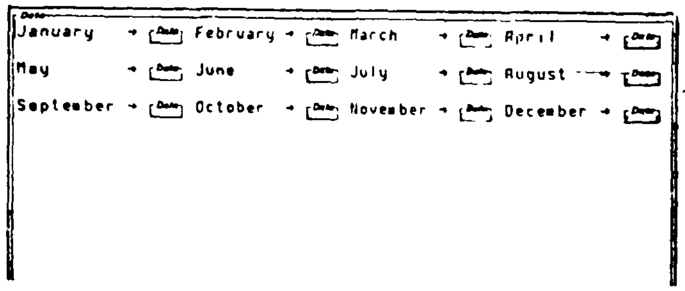


Figure 3

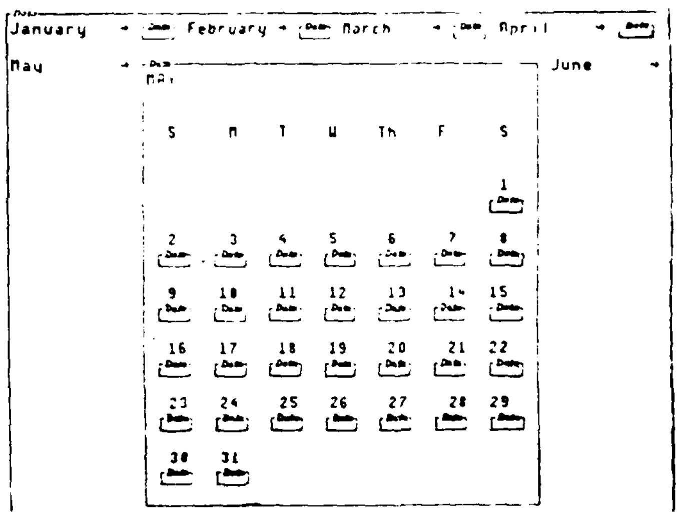
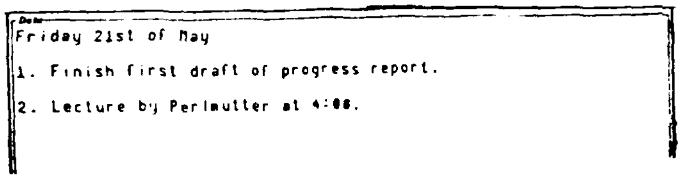


Figure 4

In Figure 3 we enter Dave's CALENDAR box. This is a collection of sub-boxes, one for each month. Figure 4 shows the MAY box expanded in place, so that we can see its contents. Boxes can be expanded or shrunk under user control, by means of the buttons on the mouse.



```
Friday 21st of May
1. Finish first draft of progress report.
2. Lecture by Perlmutter at 4:00.
```

Figure 5

Figure 5 shows us inside the box for May 21. This box contains only text, which can be edited at will. The Boxer incorporates a real-time visual editor. Any keys hit cause letters to be inserted at the cursor position in the box. In addition there is a MAKE-BOX key which inserts a box at the current cursor position. Unlike traditional, non-integrated systems, this editor is *always* present as part of the user interface, whether the user is managing data, running programs, or working with text.

3.3. The Spatial Metaphor Pervades The System

So far, we have seen that Boxer's spatial organization carries a powerful metaphor, that of "moving through" a hierarchical structure. This organization was inspired by the Spatial Data Management System [2] designed by the MIT Architecture Machine Group. But Boxer goes beyond spatial *data management*, in using the same geometric organization for all of the following hierarchies, which are treated separately in most programming systems:

- the organization of a user's programs and data according to specific applications

- the organization of shared meanings for programming language identifiers (i.e., the organization supplied by the scoping rules or block structure of a programming language)

- the organization of sharing procedures and data among different program modules[1]

- the hierarchical structure of data objects (i.e., the organization provided by arrays or lists)

[1]This is closely related to scoping of identifiers, yet distinct from it. It is supplied in Smalltalk, for example, by the class hierarchy, which is a mechanism in addition to Smalltalk's dynamically scoped identifiers. Act-I [3] supplies a similar delegation hierarchy, while maintaining lexical scoping of identifiers. The Lisp Machine uses dynamically scoped identifiers and supplies the flavor system to deal with this kind of sharing.

- the hierarchical structure of expressions within the programming language itself

3.4. Programs Can Be Constructed Concretely

In Boxer, any text appearing on the screen, whether typed by the system, typed by the user, previously executed or not, is available to be manipulated, edited, or re-executed with "do-it." This principle of "what you see is what you have" enables a mode of program construction, known as *concrete programming*, whereby the user types and executes statements one by one and then at some later time indicates that the typed statements (possibly after editing them) should be incorporated into a program. This on-the-fly programming methodology has the further integrative effect of minimizing the distinction between constructing a program and running it.

As a simple scenario of this style of programming, Figure 6 shows us working in the *Turtle* part of Boxer system, this is an environment in which we can write and execute graphics programs. The small box marked LIBRARY in the upper right hand corner is the geometry *local library* that contains definitions of symbols that are local to the TURTLE box. In this case they might be built-in procedures for manipulating a graphics cursor, together with any symbols we will define in this environment. We'll assume that the built-in graphics primitives are FORWARD and RIGHT. FORWARD causes a graphics cursor to move forward leaving a trail on the screen. RIGHT causes the cursor to rotate in place. Such a graphics cursor is called a "turtle."[2]

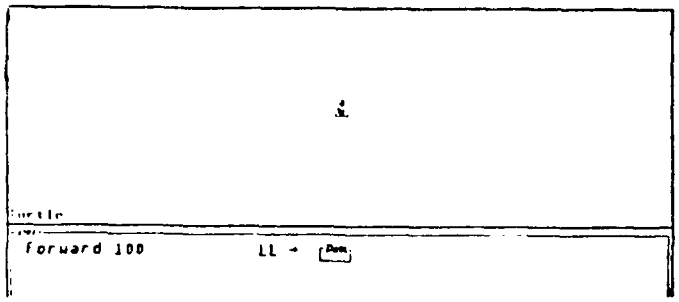


Figure 6

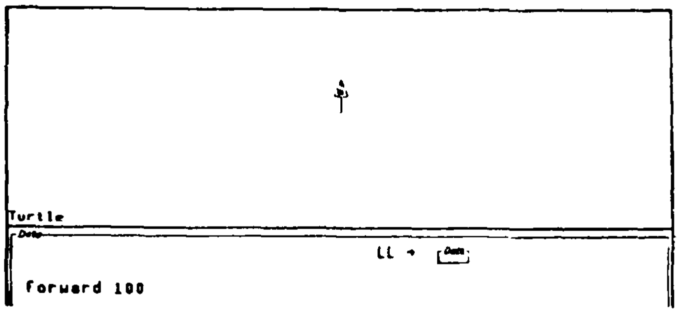
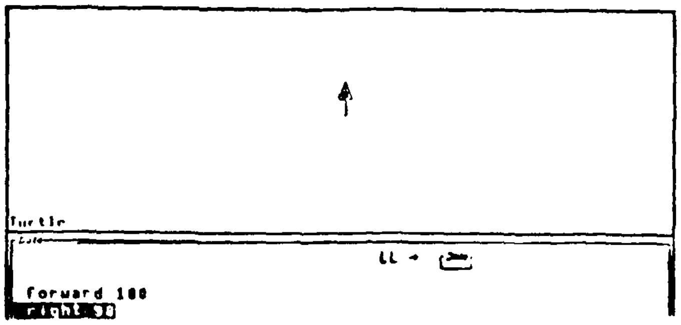


Figure 7

We can now type text to be edited and/or executed. For example, typing FORWARD 100 (Figure 6) followed by "do-it" will make the turtle cursor move forward and draw a line 100 units long. Once the text of the FORWARD command is on the screen, it can be re-executed any number of times by pointing to it and

[2]Drawing pictures by moving a cursor with FORWARD and RIGHT commands leads to a new approach to the study of geometry, called "Turtle Geometry" [4].

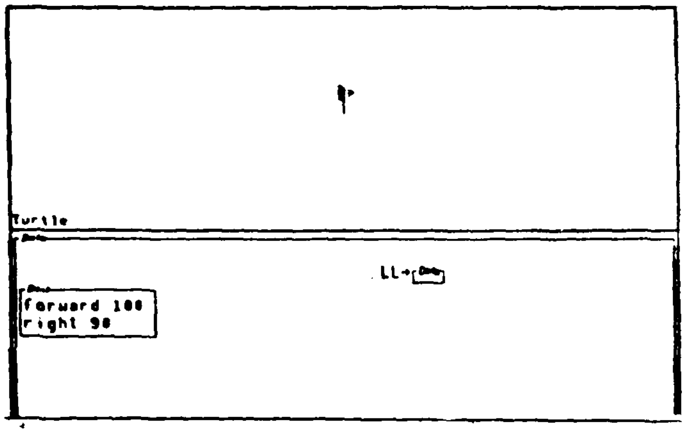
specifying "do-it." We are also free to edit any command that appears on the screen. For example, if we would rather try "FORWARD 50," all we have to do is move the cursor to the proper position and change the input.



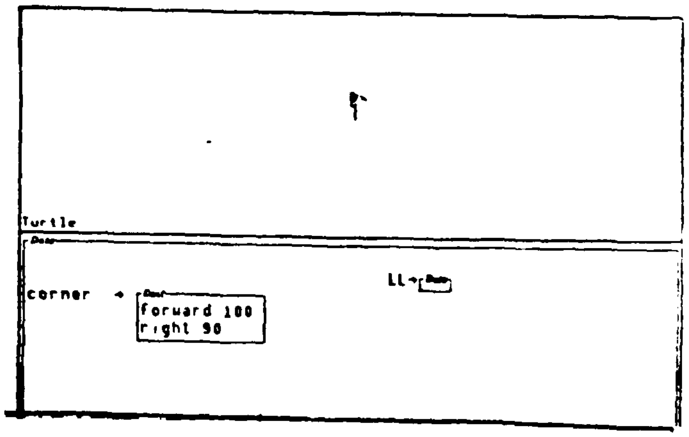
```
forward 100
right 90
```

Figure 8

Figure 8 shows us adding and executing another command, which makes the turtle rotate right 90 degrees. If we leave both commands on the screen, then we automatically obtain the functionality of a menu for issuing graphics commands. In fact, the turtle box could have been stored like this to begin with, so that a user entering the turtle environment could automatically obtain such a menu.



```
forward 100
right 90
```

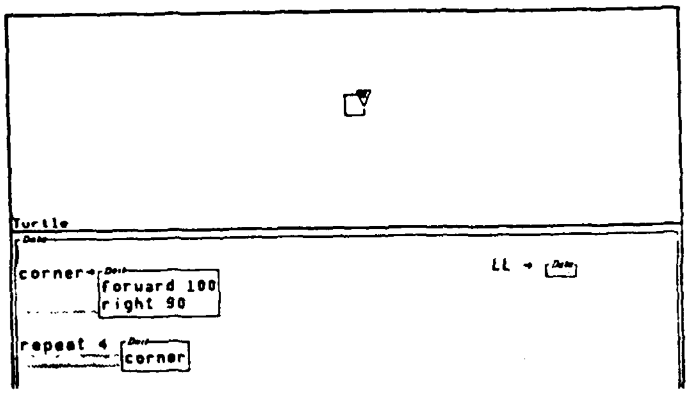


```
corner → forward 100
         right 90
```

Figure 9

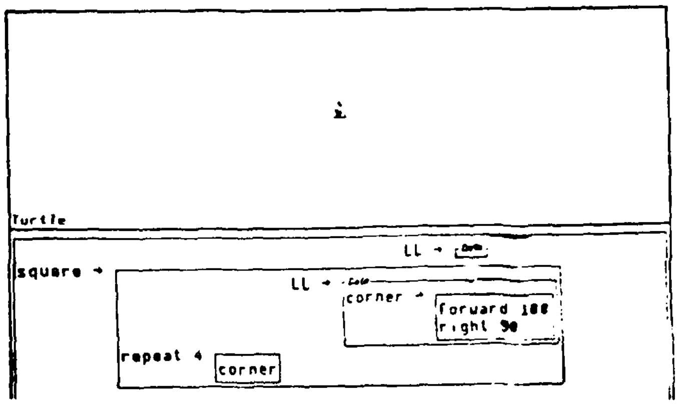
Figure 10

In Figure 9, we have grouped the commands by drawing a box around them. Now pointing to this new box and indicating "do-it" will make the turtle draw a right-angle corner. In Figure 10, we have named the box CORNER. This effectively defines CORNER as a procedure.



```
Turtle
corner → forward 100
         right 90
                                    LL →
repeat 4 corner
```

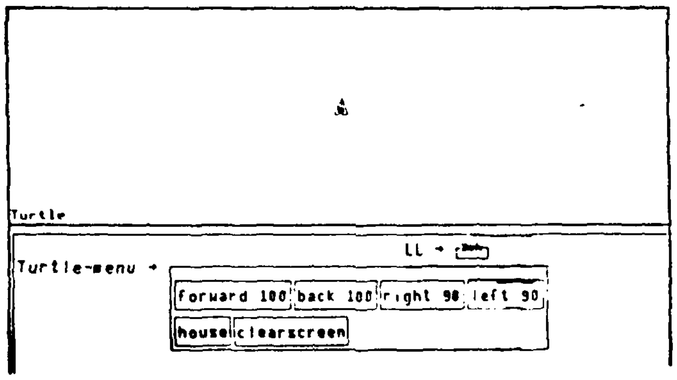
Figure 11



```
Turtle
                                    LL →
square →            LL →
                         corner → forward 100
                                  right 90
         repeat 4 corner
```

Figure 12

Figure 11 shows us drawing a square by repeating CORNER four times. Note how boxes are also used to organize the structure of the REPEAT statement, thus incorporating the syntactic grouping functionality of the BEGIN-END blocks of Algol-like languages. Figure 12 shows CORNER and the REPEAT statement grouped together to form a procedure called SQUARE. CORNER is placed in the local library for SQUARE, which effectively defines it as a procedure local to SQUARE.



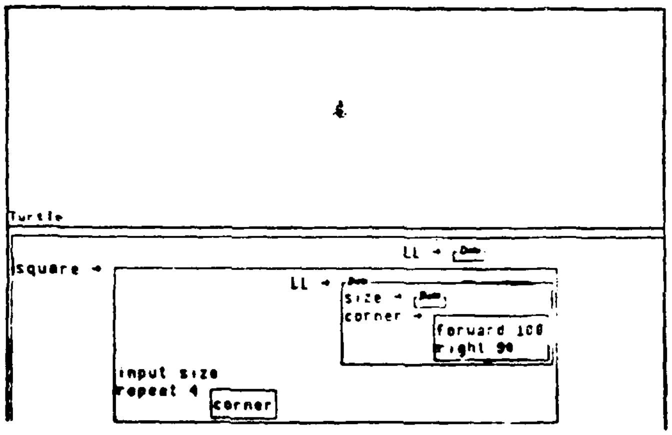
```
Turtle
                              LL →
Turtle-menu →
     forward 100 back 100 right 90 left 90
     house clearscreen
```

Figure 13

Figure 13 shows a more elaborate menu for the turtle environment. This illustrates how the system provides the menu functionality found in many novice-oriented systems. But in Boxer, a menu is no longer a special feature which a program may or may not have. Instead, the user moves freely from employing menus, to creating and editing them. He does this the same way he does anything else; to employ an already created menu, he points to his choice and presses DOIT; to create or edit a menu, he points to the part he wants to change and types in or deletes a command.

3.5. Boxes Provide Block Structure and Local Variables

Figure 12 shows SQUARE with its local procedure, CORNER. This indicates how the system provides the block structure of Algol-like languages, because the nesting of the boxes also determines the nesting of environments as far as the scoping of variables is concerned. Identifiers are searched for first within the local library of the box in which the code is being executed, then in the local library of the containing box, and so on. For instance, we could edit the SQUARE procedure so that the size of the squares drawn will be determined by a variable SIZE, as shown in Figure 14.



```
forward 100
right 90
```

```
input size
repeat 4 corner
```

Figure 14

The INPUT statement indicates that SIZE is to be an input to SQUARE, which will be installed in local library (in the box provided) when the procedure is called. Note that SIZE appears as a free variable in CORNER, which accesses it by virtue of the fact that CORNER also is located in the local environment for SQUARE.

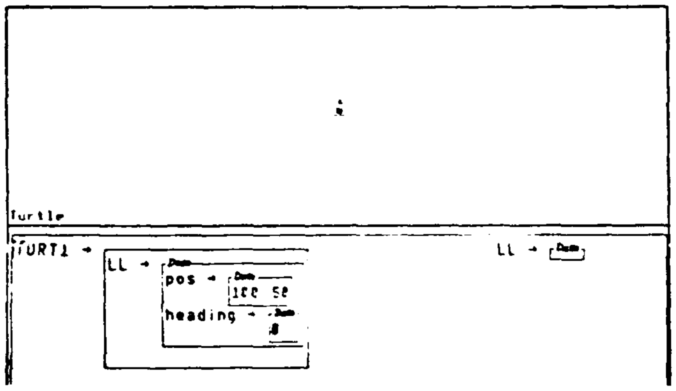
Boxer interpretation can be viewed in an "Algol Contour Model-like" manner. Identifiers are searched for in the succession of local libraries, starting with the box where reference to the identifier is made, and working outward. When a procedure is called, the box which is the procedure is copied into the *calling environment*, inputs are entered into the box's local library and the code portion of the box is executed.[3] Incidentally, this shows how the "box" representation automatically incorporates a contour model that can be used to explain the semantics of the system.

[3]Note that this implies that free variables in the procedure body will be searched for, first in the procedure's local library and then in the calling environment. Boxer is thus an dynamically scoped language rather than a lexically scoped one. One major reason for this choice is so that boxes can be used for "message passing" as explained below. Even though the language is dynamically scoped, the ability for a procedure to carry its own local library means that the usual "funarg problems" of dynamically scoped languages can in practice be avoided.

3.6. Boxes Provide Object-Oriented Programming

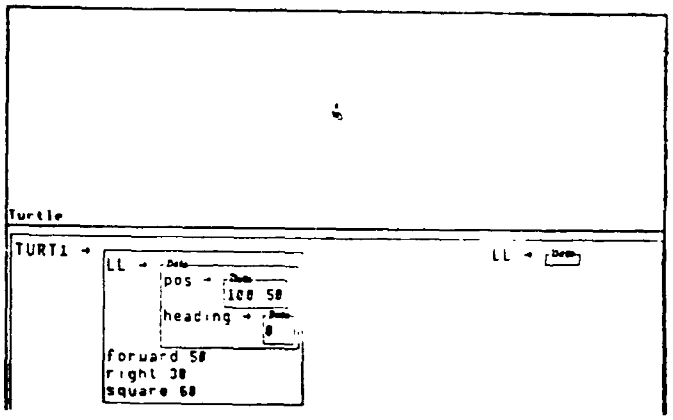
Besides serving as procedures and environments, boxes also can be used to program in an object-oriented, or "message-passing" style.

As an example, we'll implement a turtle as an object. and show how to build a system with multiple turtles. We can implement a turtle. say TURT1, as a box containing these variables in its local library. for example. by building a box in which the variables are assigned appropriate initial values as shown in Figure 15. (Note that POS, as a vector, is represented as a box containing two numbers.)



```
TURT1 →  LL → pos → 100 50
              heading → 0
                                   LL →
```

Figure 15



```
TURT1 →  LL → pos → 100 50
              heading → 0
         forward 50
         right 30
         square 60
                                   LL →
```

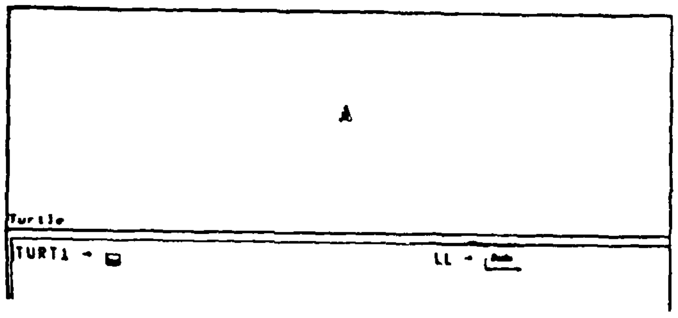
Figure 16

To make TURT1 go FORWARD, turn, or draw a square, we simply execute the appropriate commands from within the TURT1 box, as shown in Figure 16. Notice that the dynamic scoping discipline determines that the POS and HEADING variables accessed by FORWARD and RIGHT will be the ones local to TURT1.

Another way to make TURT1 move is to use the TELL command, as in

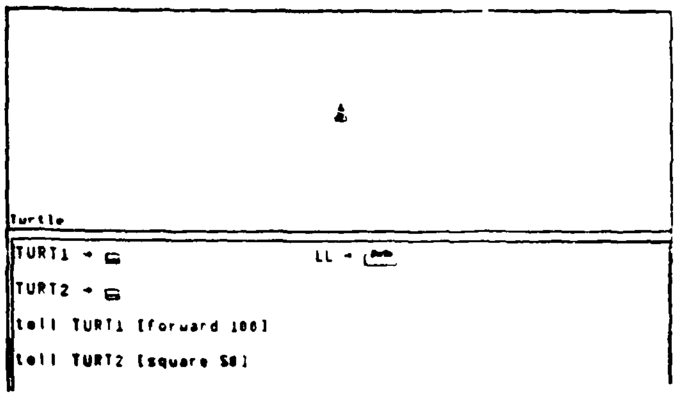
```
TELL TURT1 [FORWARD 100]
```

In general, TELL means "execute the following command within the designated box." This provides the capability that would be expressed in actor languages as "sending the object TURT1 the message FORWARD 50."



```
Turtle
TURT1 →                    LL →
```

Figure 17



```
Turtle
TURT1 →                    LL →
TURT2 →
tell TURT1 [forward 100]
tell TURT2 [square 50]
```

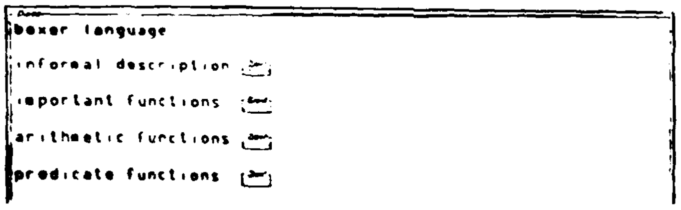
Figure 18

Of course, we could have another turtle, TURT2 that is identical to TURT1 except for its name, and could give commands to them either by entering the appropriate turtle box and typing the command, or by using TELL, as shown in Figure 18. Notice that TURT1 and TURT2 share knowledge about FORWARD, RIGHT and SQUARE by virtue of their inclusion in the TURTLE box. Alternatively, if we wanted TURT2 to execute FORWARD commands in some special way, we could do this by including the special FORWARD as a local procedure within TURT2. Then whenever we entered the TURT2 box, or used TELL TURT2, we would obtain this local FORWARD procedure. Moreover, in a situation such as

```
TELL TURT1 [SQUARE]
TELL TURT2 [SQUARE]
```

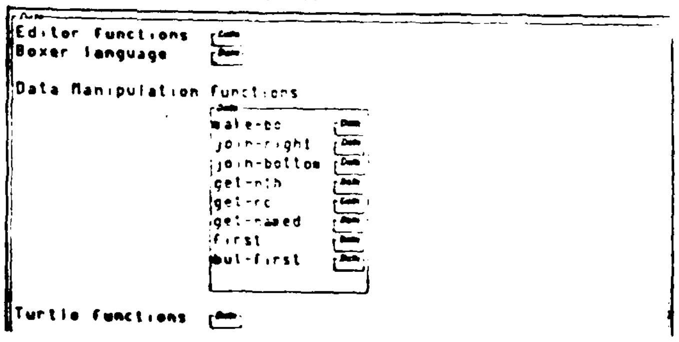
we would have TURT1 executing SQUARE using the default FORWARD procedure (i.e., the one contained in the TURTLE local library) and TURT2 executing SQUARE using its local FORWARD procedure. This example shows how the hierarchical box structure can be used to capture the "class-subclass" structure of actor languages.

3.7. Boxes Are a Natural Vehicle For System Documentation



```
boxer language
informal description
important functions
arithmetic functions
predicate functions
```

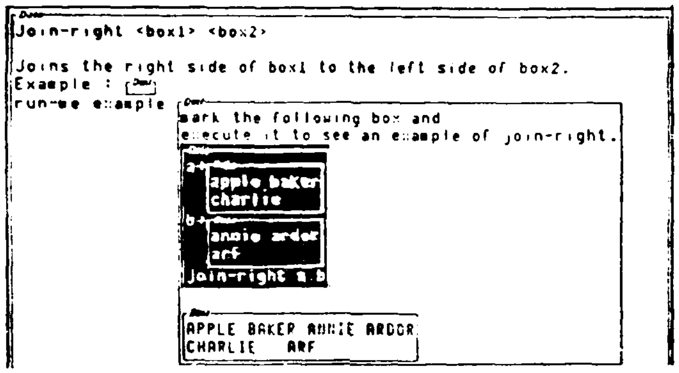
Figure 19



```
Editor functions
Boxer language

Data Manipulation functions
                          make-bc
                          join-right
                          join-bottom
                          get-nth
                          get-rc
                          get-named
                          first
                          but-first

Turtle functions
```

Figure 20

Figure 19 shows part of the on-line documentation of the Boxer system, designed and implemented by David Neves. As with any boxes, this is simply typed in as text, and the user of the documentation consults it by moving around within it. Figure 20 shows use moving into the "Important Functions" sub-box of the documentation, and expanding the box dealing with data manipulation functions.



```
Join-right <box1> <box2>

Joins the right side of box1 to the left side of box2.
Example :
run-me example
                 mark the following box and
                 execute it to see an example of join-right.
                 a←  apple baker
                     charlie
                 b←  annie ardor
                     arf
                 Join-right a b

                 APPLE BAKER ANNIE ARDOR
                 CHARLIE     ARF
```

Figure 21

Figure 21 shows the documentation for the JOIN-RIGHT command, which concatenates two boxes horizontally. (Boxer includes a full repertoire of procedures for manipulating boxes as data objects.) The documentation includes an innovation by Neves called a "run-me example." This is simply a box that the user can mark and execute in order to see an example of the command in operation. Again, the uniformity of the Boxer system allows these examples to be constructed simply as text, just as menus are constructed as ordinary text (Figure 13).

4. NEXT STEPS WITH BOXER

The implementation of the Boxer system described in the previous section is almost complete, and we expect to finish the implementation on the Lisp Machine over the summer. After that, a few missing features need to be added, most notably facilities for dealing with sharing, error handling, and interactive help. At this point we will be ready to begin sample "application" programs in Boxer, and to test it on a wider scale with novice users.

Also, within the next year we must decide upon the implementation of Boxer on another machine. The Lisp Machine is fine for developing prototypes but our system can only improve with its use by a large group of diverse people; system developers by definition do not make good novice users and we must expect to modify our ideas and our implementations according to the reactions of novices. Once the design is stable we propose to implement it on a cheaper personal computer. The primary constraints are that the machine we choose must have a bit-mapped display, a mouse (or a similarly easy to use pointing device), and substantial address space. As of now no such inexpensive machine is being sold but we expect this situation to change within the next year.

References

1. diSessa, A. "A Principled Design for an Integrated Computational Environment," to appear, 1981.

2. Bolt,R.A. Spatial-Data-Management, Massachusetts Institute of Technology, Cambridge, MA, 1979.

3. Lieberman, H. "Sharing Knowledge by Delegating Messages," MIT Artificial Intelligence Laboratory Memo, Cambridge, MA, 1979.

4. Abelson, H. and diSessa, A. Turtle Geometry: The Computer as a Medium for Exploring Mathematics, MIT Press, Cambridge, MA, 1981.

Publications

1. Abelson, H. and diSessa, A. Turtle Geometry: The Computer as a Medium for Exploring Mathematics, MIT Press, Cambridge, MA, 1981.

2. diSessa, A. "An Elementary Formalism for General Relativity," American Journal of Physics (May 1981).

3. diSessa, A. "The Computer and Mathematical Experience," Proceedings of the Fourth International Congress on Mathematics Education, in press.

4. diSessa, A. "Unlearning Aristotelian Physics: A Study of Knowledge-Based Learning," to appear in Cognitive Science.

5. diSessa, A. "Phenomenology and the Evolution of Intuitions," to appear in *Mental Models*, D. Gentner and A. Stevens, (Eds.) Lawrence Erlbaum Press.

Talks

1. diSessa, A. "The Role of Experience in Learning Physics," Symposium on Mental Models at the Annual Cognitive Science Society Meeting, San Francisco, CA, August 19-20, 1981.

2. diSessa, A. "Phenomenology and the Evolution of Intuition," University of Rochester, Rochester, NY, October 30, 1981.

3. diSessa, A. "Learnability Principles for the Design of Computational Environments," Xerox Palo Alto Research Center, Palo Alto, CA, anuary 21,1982.

4. diSessa, A. "The Role of Experience in Learning and Knowing Physics," invited address at the annual Joint Meeting of the American Physical Society--American Association of Physics Teachers, San Francisco, CA, January 26, 1982.

5. diSessa, A. "Computers in Math and Science Education: The Rebirth of Exploration?" Columbia University, Teachers College, New York, April 12, 1982.

6. diSessa, A. "Turtle Geometry--What's It About" and "Advanced Topics in Turtle Geometry: Physics and Differential Geometry," Sessions at the Third Annual ECOO Conference on Computers in Education, Toronto, Canada, April 29, 1982. At the same conference "The Future of Computers in Education" (panelist).

7. diSessa, A. "Philosophical Issues in the Design of Computer Languages," a discussion with Adele Goldberg and Ken Iverson contrasting Logo, Smalltalk and APL, Canadian ACM, SIG APL Meeting, April 28, 1982.

8. diSessa, A. Workshop on Intuitive Physics, University of Rochester, Rochester, NY, May 17-29, 1982.

9. Neves,D. "Boxer: An Integrated System for Novice Computer Users," at the Workshop of Very High Level Languages, University of Pennsylvania, Philadelphia, PA, February 1982.

FUNCTIONAL LANGUAGES AND ARCHITECTURE GROUP

Academic Staff

Arvind, Group Leader

Research Staff

R. Thomas

Graduate Students

R. Iannucci	K. Pingali
V. Kathail	W. Seltzer
J. Pineda	

Undergraduate Students

S. Gray	S. Heller
D. Perich	J. Rodriguez
K. Suh	

Support Staff

K. Warren

1. INTRODUCTION

The Functional Languages and Architectures group is studying new computer structures to exploit the parallelism which is easily found in many functional programs. Our approach in studying parallelism is based on a highly dynamic interpreter for dataflow graphs called the U-interpreter [1] Since we believe the success of a general-purpose multiprocessor computer depends on its effective programmability and its efficient utilization of resources we are concerned not only with hardware issues but also with associated system problems such as high level language support, communication requirements, and efficient distribution of workload over the machine. We anticipate that a variety of approaches, languages, etc. will be applied towards resolving these problems depending on the application context.

We are continuing our work in developing the high level dataflow language Id and compilers for it to generate machine code for a prototype dataflow machine. We have started the process of specifying the functionality of each component of the prototype machine in enough detail to facilitate hardware implementation.

Our group, together with Computation Structures Group organized the ACM Conference on Functional Programming Languages and Computer Architecture in Portsmouth, NH (October 18-22, 1981). Approximately 200 people from all over the world participated in the conference to hear 26 papers and 2 panel discussions on various functional languages, logic programming, and reduction and dataflow architectures for executing side-effect free languages. An edited transcript of the questions and answers that followed every presentation and the two panel discussions is under preparation and will soon be published as a technical report. Arvind and Tilak Agerwala of IBM also edited a special issue of IEEE Computer Magazine on Dataflow Systems (February 1982).

2. LANGUAGE RELATED WORK

Arvind, D. Brock, and K. Pingali have investigated the extension of Backus' FP language to permit user-defined higher order functions. Although the importance of higher order functions is not always fully appreciated, we feel that it is imperative that functional languages support them. Backus' FFP language has this power, but it comes at a price, i.e., FFP is not extensional. We have extended Backus' language FP to support higher order functions without disturbing the mathematically useful properties of FP, most importantly, its algebraic identities [5] whose variables range over all FP programs. The only price we pay for this extension is the relative complexity of the semantic domain of the extended FP in comparison with the semantic domain of FP.

As part of his master's thesis, K. Pingali studied the problem of implementing streams so that stream programs are safe for data driven evaluation. For a simple stream language L_0, which is a subset of Id, he developed a scheme for transforming any program in L_0 into an L_0 program that is safe for data driven evaluation. The transformation scheme is also relevant for demand driven evaluation because the transformed scheme generates far fewer suspensions than the source program. The language L_0 has the usual stream operators first, rest, cons, and in addition, it allows other operators that are *one-in-one-out*. We are investigating the extension of the techniques developed to a language L_1 which includes the *filter* operator that does not have the one-in-one-out property. Even though this work is being pursued in the context of dataflow, it has direct implications for the general problem of efficient code generation from functional languages. The techniques being developed will substantially improve the efficiency of current lazy interpreters for functional languages.

The Id compiler project to translate a subset of Id (i.e., Id without streams and managers) to Lisp was completed by V. Kathail and K. Pingali. This Id compiler is part of IdSys (an abbreviation for Id System) which is essentially an environment for writing, compiling, debugging and running Id procedures. IdSys currently runs on the XX system (DEC-20) in the Laboratory for Computer Science. A manual for using IdSys is also available [9] The compiler was used by students in the dataflow course 6.847, and is currently being used to write a statistical program package by George Gedeon in the center for advanced engineering studies.

A compiler to translate Id (excluding managers) into dataflow graphs has also been completed. It generates an abstract parse tree which is used in the following pass to determine the rarity of expressions, the variables used in an expression, and the point at which each variable is defined. Most of the semantic error analysis is done during this pass. A subsequent pass over the parse tree then generates dataflow graphs. The graphs generated by the compiler differ in some respect from the ones described in [1] because of the special treatment given to I-structures [4].

Ki Suh is working on the program which will translate dataflow graphs into the actual instructions for the tagged-token machine as described in This program will form the final phase of the code generation process, and will be integrated with the rest of the compiler. Work on the detection of I-structures, i.e., to determine when a general structure can safely be assumed to be an I-structure, and certain optimizations like constant propagation, dead code elimination, code-block merging, etc. is in progress.

3. A DATAFLOW ARCHITECTURE AND PROTOTYPE IMPLEMENTATION

During the last year our group has made significant progress in finalizing the design of the 64 processor prototype dataflow machine. The work concentrated on specifying the instruction set and a hardware design of the processing element to be built in the laboratory by the end of 1983. We have now begun the process of specifying the major subsystems of the prototype machine in terms of a Pascal program which implements the functionality of each subsystem. These programs will serve as detailed and unambiguous specifications for these subsystems and will serve as input to subsequent steps in prototype implementation as well as communicating our design to others. Pascal was chosen for this task because it is widely known, transportable, and is readily adaptable to the uses we expect of the functional specifications such as software simulation, transliteration to microcode, and eventually specification for custom VLSI chips. R. Thomas is coordinating the work on these specifications and has completed the specifications for the I-structure controller.

3.1. Instruction Set Specification

Arvind and V. Kathail had specified the outline of an instruction set to handle procedures, loops and I-structures in the previous year [2]. However, the instruction set needed many more details concerning the format of values and tokens, and addressing mechanism. Arvind and R. Iannucci have completed these specifications and in the process have also made changes in the format of the instructions [3] A better scheme than the one given in [2] for mapping U-interpreter generated activity names onto fixed-size hardware tags has also emerged from discussions within the group. The new tagging scheme (as suggested by V. Kathail) and the instruction set are reported in [3]. In the following we describe the salient changes and new features of the instruction set.

Constant Specification - Constants are now associated only with loops and stored in program memory instead of I-structure memory. This change was made to avoid an indefinite wait by the ALU in fetching a constant from the local I-structure storage. Special treatment of constants in procedures has been eliminated because it seemed to offer no practical advantage.

Instruction Format - Instructions can have as many as three operands. In case of a three operand instruction, one operand must be a constant. Inclusion of a field known as the *disposition* of an operand allows any input to be ignored and thus, to be used as a trigger. Destination fields in instructions are independent of physical location to permit dynamic loading of procedures without modifying the code.

Generation of Tags - An algorithm for generating the tag and the PE number for an output token from the input tag and destination information is specified. Previously a maximum of 16 concurrent procedure and loop activations were permitted in a PE. The new scheme allows up to 16 concurrent invocations of each individual code block. In the case of procedure code blocks, the number of concurrent activations can be even larger because it is possible to use the iteration field to distinguish between many activations of the same procedure. The new tagging scheme still uses only 28 bits (4 for color, 16 for the instruction address within a PE, and 8 for iteration number) but is able to use all 2^{28} names for naming activities.

New Instructions - A complete set of instructions has been included to manipulate token values at the bit and byte level.

Although the specification of the instruction set is almost complete, some problems remain in the generation of code for data structure operations. Generation of I-structure instructions requires type information which either is not available in Id or is difficult to derive. Also, we plan to use I-structures to implement streams but the translation of streams into I-structure instructions has not been worked out yet. A complete set of instructions to implement managers can be specified only after stream implementation is better understood.

3.2. Alternatives for Hardware Implementation of the Prototype

We considered a large variety of strategies for implementing a prototype dataflow machine. Although we plan to eventually implement our design with custom VLSI chips we have selected a lower risk implementation technology for the first 64 processor demonstration machine which is to be constructed using AMD 2903 and other off-the-shelf components. Other strategies we considered were:

1) One or more off-the-shelf single-board microprocessors (M68000) to emulate each PE and a "migration strategy" to incrementally incorporate custom VLSI chips;

2) One 8-bit EPROM programmable microprocessor chip (M68701) per PE subsection;

3) Three microprocessor chips on a single board to functionally emulate a PE;

4) One AMD 29116 and other off-the-shelf chips on a board to implement each subsection of a PE;

5) Custom gate array implementation in cooperation with an industrial partner.

The analysis of all but the last two of these approaches was reported by R. Iannucci [6]. We chose the micro machine based on the AMD 2903 and other off-the-shelf chips as the most realistic strategy for a group of our size to accomplish while having very good performance, flexibility, and applicability for design of the eventual custom VLSI implementation.

3.3. The Micro Machine Prototype

Each PE of this machine will be a fairly high-performance 32-bit micro programmable machine (containing at least 256K bytes of dynamic RAM). R. Iannucci and R. Thomas designed this machine [6] and its micro assembler so that it supports efficient implementation of the dataflow machine instruction set. For example, the low-level structure of the micro machine relies on microcode-level context switching in the spirit of the Xerox Alto [11] and Dorado [7]. (Such hardware supported context switching is extremely important for efficient implementation of the rather fine-grained parallelism of our dataflow PE.) Furthermore, the data paths, memory, and arithmetic elements are sufficiently wide (32 bits) so that no convoluted code is needed to manipulate normal data objects (*e.g.*, floating-point numbers, memory addresses). Automatic alignment circuitry and length/boundary hardware eliminate problems in manipulating objects smaller than 32 bits. The control storage subsystem implements a very sophisticated yet simple branching mechanism to allow for 2-way, 4-way, 8-way, ..., or 256-way branching on any data object. Multi way branching can be performed on contiguous and noncontiguous bits within the same eight-bit byte. And finally, the maintenance subsystem contains a microprocessor which is separate from the main data paths; it facilitates initialization, testing, debug, and control of the micro machine. The primary memory (constructed from 64K bit or 256K bit dynamic RAMs) will be error corrected to further enhance the machine's reliability.

The syntax of micro instructions to be accepted by the micro assembler is equational, with limited use of keywords for branching and other controls. One of the major issues in the design of such a micro assembler is the problem of assigning micro instructions to control storage addresses. Due to the regular nature of micro instruction addressing (we have not used a blocked control storage structure), this is a containable problem. Assignment can be done quite effectively as a three step process:

1) Select cells for the targets of all multi way branches starting with 256-way, then 128-way, etc.;

2) Select cells for all fixed odd/even pairs:

   - 2-way branching using the odd/even select mechanism;

- Subroutine linkages;

- Micro instructions using immediate constants and their successors;

3) Assign the remaining micro instructions freely to the remaining cells.

3.4. System Level Support

Implementation of a usable prototype machine involves a great deal of support software and hardware for tasks such as down loading code, debuggers, test jigs, etc. R. Thomas has implemented a flexible debugging system which can be easily configured for minimum M68000 systems such as the service processor on the micro machine. Two UROP students directed by R. Thomas and a bachelors thesis student project have also contributed in this area. J. Rodriguez wrote a program for transferring information (using a 9600 baud serial link) between a M68000 microprocessor and a Data I/O System 19 PROM programmer. This program makes it convenient to down load code or data from XX to a M68000 microprocessor and subsequently to program it into EPROMs, PROMs, or PAL chips for use in the prototype. S. Gray worked on supervisor calls which would allow sophisticated debuggers and I/O handlers to be constructed on the M68000 service processor of the micro machine. And D. Perich worked on a parallel-port interface which will be used to connect the prototype dataflow machine and a VAX11-750 host computer.

3.5. VLSI Implementation

Since VLSI is the most suitable technology for implementing dataflow multiprocessors, we believe that it is imperative to develop and maintain expertise in VLSI design. We view this as long term, high payoff research in which characteristics unique to VLSI may be exploited in future dataflow architectures.

Interconnection Networks - In conjunction with his VI-A Co-op assignment with Bell Laboratories, Wayne Seltzer has completed the VLSI design of a communications module for the tagged-token dataflow architecture [10] The CMOS integrated circuit of approximately 40,000 transistors functions as an 8x8 packet router. An asynchronous protocol allows a number of these chips to be interconnected in a variety of topologies to provide an arbitrarily large packet-switched communications network. Packets, of up to 128 bits, serially enter one of the router's eight input ports. A routing address in the packet is used to determine the output port destination. The router maintains flow control through the network with on-chip buffering and internally arbitrates output port contention.

Waiting-Matching - The waiting-matching section in each PE of our prototype machine brings together operands for subsequent execution by associatively matching the tags carried on input tokens. Towards providing a low-cost implementation for this function, J. Pineda and R. Thomas specified [8] the functionality of a fully associative content-addressable memory (CAM) which J. Pineda then designed and implemented as a NMOS VLSI chip. The chip provides sixty-four 32-bit words and can be cascaded to extend the number of words. Either a 16-bit or 32-bit data bus can be used depending on packaging (currently the chip is packaged in a 40-pin DIP providing a 16-bit data bus; 56 pins would be needed for a 32-bit bus). The chip was fabricated in the spring of 1982 by Bell Labs; J. Pineda has tested a large percentage of the chip using facilities at Prime Computer including a Fairchild Sentry tester and probe station. These tests revealed three design errors: two missing wires and one logical error. By using micro probes to substitute for the missing wires, most of the remainder of the chip has been determined to be functional. A corrected version of the chip was sent for fabrication in June 1982.

References

1. Arvind, Gostelow, K.P. and Plouffe W. "An Asynchronous Programming Language and Computing Machine," Technical Report TR 114a, Department of Information and Computer Science, University of California - Irvine, Irvine, California, December 1978.

2. Arvind, and Kathail, V. "A Multiple Processor Dataflow Machine that Supports Generalized Procedures," Proceedings of the 8th Annual Symposium on Computer Architecture, Minneapolis, MN, May 1981.

3. Arvind, and Iannucci, R. A. "Instruction Set Definition for a Tagged-Token Data Flow Machine," Memo 212, Computation Structures Group, MIT Laboratory for Computer Science, Cambridge, MA, December 1981 (revised May 1982).

4. Arvind and Thomas, R.E. "I-Structures: An Efficient Data Type for Functional Languages," MIT/LCS/TM-178, MIT Laboratory for Computer Science, Cambridge, MA, September 1980, (revised October 1981).

5. Backus, J. "Can Programming be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs," CACM, 21,8 (August 78), 613-641.

6. Iannucci, R.A. "Implementation Strategies for a Tagged-Token Data Flow Machine," Memo 218, Computation Structures Group, MIT Laboratory for Computer Science, Cambridge, MA, June 1982.

7. Lampson, B. W., and K. A. Pier "A Processor for a High-Performance Personal Computer," Xerox Palo Alto Research Center, Palo Alto, CA, January 1981

8. Pineda, J. and Thomas, R. "Preliminary Functional Specification of a CAM for use in a Tagged-Token Dataflow Processor," unpublished note, December 11, 1981.

9. Pingali, K. and Kathail, V. "IdSys Manual," Memo 211, Computation Structures Group, MIT Laboratory for Computer Science, Cambridge, MA, December 1981.

Seltzer, W. A. "A Communications System for a Multiprocessor Dataflow Computer Architecture," S.M. and S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, (expected August 1982).

10. "ALTO: A Personal Computer System - Hardware Manual," Xerox Palo Alto Research Center, Palo Alto, CA, May 1979.

Publications

1. Agerwala, T. and Arvind "Guest Editorial -- Special Issue on Data Flow Systems" Computer Magazine, (February 1982) 10-13.

2. Arvind and Brock, J.D. "Streams and Managers," Memo 217, Computation Structures Group, MIT Laboratory for Computer Science, Cambridge, MA, To appear in Springer-Verlag Lecture Notes on Computer Science series.

3. Arvind and Gostelow, K.P. "The U-interpreter," Computer Magazine, (February 1982) 42-49.

4. Arvind and Iannucci, R. A. "Instruction Set Definition for a Tagged-Token Data Flow Machine," Memo 212, Computation Structures Group, MIT Laboratory for Computer Science, Cambridge, MA, December 1981 (revised May 1982).

5. Arvind and Pingali, K. "Safe Data-driven Evaluation," Proceedings of the Symposium on Functional Languages and Architecture, Goteborg, Sweden, June 1981 (revised April 1982).

6. Arvind and Thomas, R. E. "I-structures: An Efficient Data Type for Functional Languages," MIT/LCS/TM-178, MIT Laboratory for Computer Science, Cambridge, MA, June 1980 (revised October 1981).

7. Iannucci, R.A. "Implementation Strategies for a Tagged-Token Data Flow Machine," Memo 218, Computation Structures Group, MIT Laboratory for Computer Science, Cambridge, MA, June 1982.

8. Pineda, J. and Thomas, R. "Preliminary Functional Specification of a CAM for use in a Tagged-Token Dataflow Processor," unpublished note, December 11, 1981.

9. Pingali, K. and Kathail, V. "IdSys Manual," Memo 211, Computation

Structures Group, MIT Laboratory for Computer Science, Cambridge, MA, December 1981.

Theses Completed

1. Perich, Daniel N., "A Parallel Interface Between a PDP-11/40 and a Motorola 68000," M.S. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

Theses in Progress

1. Pingali, Keshav, "Streams in Applicative Languages," MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected August 1982

2. Seltzer, Wayne A. "A Communications System for a Multiprocessor Dataflow Computer Architecture," S.M. and S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected August 1982.

Talks

1. Arvind, "Safe Data-driven Evaluation" Workshop on Functional Languages and Computer Architecture, Gothenberg, Sweden, June 1981.

2. Arvind, "A Dataflow Architecture with Tagged tokens," I.I.T., Kanpur, INDIA, July 2, 1981.

3. Arvind, "A Dataflow Architecture with Tagged Tokens," University of Wisconsin, Madison, Wisconsin, September 24, 1981.

4. Arvind, "A Dataflow Architecture with Tagged Tokens," Memorial University, St. John, New Foundland, Canada, November 30, 1981.

5. Arvind, "A Dataflow Architecture with Tagged Tokens," MIT Laboratory for Information and Decisions, Cambridge. MA February 23, 1982.

6. Arvind, "A Dataflow Architecture with Tagged Tokens," University of Rhode Island, RI. March 3, 1982.

7. Arvind. "A Dataflow Architecture with Tagged Tokens," Intermetrics Seminar, Cambridge, MA March 10, 1982.

8. Arvind, "In praise of Functional Languages and Dataflow Architectures," Workshop on Language Issues in High Speed Scientific Computing, Glenaden Beach, Oregon, March 17, 1982.

9. Arvind, "A Dataflow Architecture with Tagged Tokens," Intel Corporation, Aloha, Oregon, March 19, 1982.

10. Arvind, "A Dataflow Architecture with Tagged Tokens," MIT-IBM Workshop, Lenox, MA, April 15, 1982.

11. Arvind, "A Dataflow Architecture with Tagged Tokens," Workshop on Parallel Processing, Courant Institute , New York, April 23, 1982.

12. Arvind, "A Dataflow Architecture with Tagged Tokens," Siemens, Munich, May 4, 1982.

13. Arvind, "A Dataflow Architecture with Tagged Tokens," Gesellschaft fur Mathematik und Datenverarbeitung Mbh, Bonn, May 6, 1982.

14. Arvind, "A Dataflow Architecture with Tagged Tokens," Kiel University, Kiel, Germany, May 7, 1982.

15. Arvind, "A Dataflow Architecture with Tagged Tokens," MIT-VLSI Research Review, May 17, 1982.

16. Iannucci, R., "Dataflow Computer Architecture: An Introduction," IBM Glendale Laboratory, Endicott, NY, January 6, 1982.

17. Pingali, K. and Kathail, V., "A Dataflow Instruction Set Which Supports Generalized Procedures," Workshop on Instruction Set Design and Code Generation for Data-Driven Computing Systems, July 6-8, 1981.

18. Thomas, R. "An Incremental Hardware Strategy for Implementing a Prototype Dataflow Machine," Workshop on Instruction Set Design and Code Generation for Data-Driven Computing Systems, July 6-8, 1981.

INFORMATION MECHANICS

Academic Staff

E. Fredkin, Group Leader

Research Staff

T. Toffoli
G. Vichniac

Graduate Students

R. Giansiracusa N. Margolus

Undergraduate Students

R. Fearing
A. Hofmann
W.L. Lee

Support Staff

R. Hegg

1. CONSERVATIVE LOGIC AND REVERSIBLE COMPUTING

One of the goals of conservative logic is to explore ways of realizing virtually nondissipative computing. To this purpose, it is necessary to achieve a very close match between the logical structure of the desired computation and the structure of the physical computer that should carry it out.

In previous years, we had shown that the "thermodynamical limit" to computation—characterized by Landauer in a well-known paper—does not apply to computation based on reversible primitives, and that such primitives are sufficient for constructing a universal computer. We also showed that such reversible primitives admit of a simple classical-mechanical realization.

1.1. Quantum-mechanical Computation

The next step in this program is to arrive at a quantum-mechanical realization of reversible computation. In the past year, we have made much progress in understanding under what conditions it is possible to make a quantum-mechanical system carry out in an exact way deterministic, computationally-universal processes in a nondissipative way. In fact, we have devised a general method for constructing a quantum-mechanical Hamiltonian describing a general-purpose computer. This method constitutes a great simplification with respect to some recent work by Benioff in the same area.

We are collaborating with Richard Feynman in identifying specific quantum-mechanical effects capable of realizing universal, reversible primitives such as the Fredkin gate or the Interaction gate.

1.2. SQUARELAND

We have devoted much time and effort to the design, construction, and testing of SQUARELAND, a high-performance machine dedicated to the simulation of cellular automata.

In this machine. the new state of the cellular-automaton array is computed in a conventional manner from the old state through sequential scanning, but all operations—including access to a cell's neighbors—are pipelined. Thus the local transition function can be computed with absolutely no overhead. In this way, with low-power Schottky TTL logic one can achieve a scan rate of 80 nsec per cell. This speed is amply sufficient to permit one to slave the scanning of the array (256x256 = 65,536 cells) to the scan rate of an ordinary CRT monitor (60 frames/sec), thus bypassing the need for a separate display memory. The present version can handle up to eight bit-planes, corresponding to up to 256 states per cell.

Double-buffering and an original addressing scheme greatly reduce memory-access bandwidth, and allow one to use slow and inexpensive memory for main storage.

In conclusion, SQUARELAND permits one to view in real time the behavior of cellular automata at a rate and a resolution comparable to that of a Super-8 movie. A number of original t^chnical solutions permit one to achieve this performance using perhaps $200's worth of readily available IC chips.

We have started using SQUARELAND for gaining an intuitive feeling on the behavior of a variety of cellular automata, and for testing a number of working hypotheses. This activity is very similar to experimental research in the natural sciences, but is directed at studying the properties of "artificial" universes whose rules can be arbitrarily specified. Using SQUARELAND, we plan to isolate and study in a stylized way specific features of physics from a computational viewpoint.

1.3. Discrete Computational Prototypes for Physics

Our work on the physics of computation was complemented by research on computational prototypes for physics. We have studied questions concerning the thermodynamics of discrete systems, computational models of quantum mechanics, models for the EPR paradox based on local interactions, the possibility of defining a simplectic (viz. Hamiltonian) dynamics for discrete systems. We have done much groundwork in the hope of arriving at a reasonable extension to discrete dynamics of Noether's theorem (which characterizes the connection between symmetries and conservation laws in continuous physical systems).

Our group organized an Information Mechanics Workshop, held in the second half of January 1982 in the British Virgin Islands, with the participation of a number of distinguished physicists and computer scientists. In this workshop, particular emphasis was given to the connections between abstract computation and theoretical physics.

1.4. Computation and Lattice Dynamics Seminars

We have collaborated with Hyman Hartman in organizing biweekly seminars on Computation and Lattice Dynamics. There are a number of homogeneous physical systems with local interactions (crystals, spin-glasses, doped crystals, etc.) whose analysis brings very closely together concepts of physics and of computation. In fact, such systems can be thought of as "accidental" (i.e., naturally occurring) parallel computers, some of them capable of performing amazing distributed-processing feats.

In these seminars, discussions were centered on presentations offered by members of our group as well as a number of physicists from MIT's Center for Theoretical Physics, Harvard, Boston University, and Northeastern.

2. SEMI-INTELLIGENT CONTROL

Semi-intelligent control is an original approach to the problem of controlling machinery through the use of a distributed network of microprocessors. This approach stresses the use of local, uniform, and redundant information to drastically reduce bandwidth, and the systematic use of look-up tables (rather than analytic methods) to build up flexible learning and performance in control tasks.

In the past year we have continued work on a number of pilot activities in this area. Having completed construction of the hinge system (a very simple, man-sized robot which should be able to stand up, balance itself, and walk in hops), we have started exercising it, first with manual control and then by feeding to it canned strings of commands (with no feedback). We plan to compare the behavior of the actual hinge robot with that of the simulation that we have been running on a LISP machine, in order to adjust the simulator's parameters and eventually drive the robot by means of the same programs that now run the simulation. To this purpose, we will equip the hinge robot with sensors capable of producing adequate feedback.

A student has just completed another project in this area, dealing with simple methods of recognizing objects by acoustic means. The project is centered on the POLAROID sonar pulser/detector, and uses a microprocessor to drive the sensor and perform the lowest level of data acquisition and analysis.

Publications

1. Fredkin, E. "Digital Information Mechanics," transcripts of a talk given to the Information Mechanics Workshop, British Virgin Islands, January 1982, now being revised and expanded into a technical memo.

2. Fredkin, E. and Toffoli, T. "Conservative Logic," International Journal of Theoretical Physics, 21 (1982), 219-253.

3. Toffoli, T. "Physics and Computation," International of Journal Theoretical Physics, 21 (1982), 165-175.

4. Toffoli, T. "SQUARELAND: A High-performance Cellular-automaton Machine," to appear.

Theses Completed

1. Payton, D. "A Predictive Learning Control System for an Energy Conserving Thermostat," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, August 1981.

Theses in Progress

1. Giansiracusa, R. "Adaptive Multilevel Modeling of Complex Dynamical Systems," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1982.

2. Margolus, N. "Physics and Computation," Ph.D. dissertation, MIT Department of Physics, Cambridge, MA, expected June 1983.

Talks

1. Fredkin, E. "Digital Information Mechanics," Information Mechanics Workshop, British Virgin Islands, January 18-29, 1982.

2. Fredkin, E. "Information and Physics," concluding address at the Livermore Laboratory Conference on High-Speed Computing, Gleneden Beach, Oregon, March 16-18, 1982.

3. Margolus, N. "Energy and Entropy in Cellular Automata," Computation and Lattice Dynamics Seminars, MIT Cambridge, MA, October 1981.

4. Margolus, N. "The Role of Energy and Entropy in Reversible Computation," Information Mechanics Workshop, British Virgin Islands, January 18-29, 1982.

5. Margolus, N. "A Local Quantum Mechanical Description of Reversible Cellular Automata," Information Mechanics Workshop, British Virgin Islands, January 18-29, 1982.

6. Toffoli, T. "Conservative Logic," invited talk at GMD, St. Augustin, West Germany, July 1981.

7. Toffoli, T. "Semi-intelligent Control," invited talk at GMD, St. Augustin, West Germany, July 1981.

8. Toffoli, T. "Nondissipative Computation," Computer Science and Systems Seminar, University of Bridgeport, October 1981.

9. Toffoli, T. "SQUARELAND: A Practical Tool for Viewing the Thermodynamics of Crystals and Lattice Systems," Computation and Lattice Dynamics Seminars, MIT, Cambridge, MA, October 1981.

10. Toffoli, T. "How Linear Systems Can do Universal Computation: Two Approaches," Information Mechanics Workshop, British Virgin Islands, January 18-29, 1982.

11. Toffoli, T. "Cellular Automata and Physics," Information Mechanics Workshop, British Virgin Islands, January 18-29, 1982.

12. Vichniac, G. "Simulating Lattice Systems With Cellular Automata," Information Mechanics Workshop, British Virgin Islands, January 18-29, 1982.

13. Vichniac, G. "Reversible Difference Equations and Limit Cycles," Information Mechanics Workshop, British Virgin Islands, January 18-29, 1982.

14. Vichniac, G. "The Copenhagen Interpretation of Quantum Mechanics," Information Mechanics Workshop, British Virgin Islands, January 18-29, 1982.

15. Vichniac, G. "Vers le Lagrangien de l'ordinateur," Physics Colloquium, University of Montreal, Montreal, Quebec, April 16, 1982.

MESSAGE PASSING SEMANTICS

Academic Staff

C. Hewitt, Group Leader

Research Staff

H. Lieberman

Graduate Students

G. Barber
E. Ciccarelli
S.P. de Jong
W. Kornfeld
P. Koton

Undergraduate Students

B. Pines
D. Theriault

Support Staff

J. Jones

Visiting Scientists

G. Attardi
M. Simi

1. INTRODUCTION

An important skill in programming is being able to *visualize* the operation of procedures, both for constructing programs and debugging them. *Tinker* is a programming environment for Lisp that enables the programmer to "see what the program is doing" while the program is being constructed, by displaying the result of each step in the program on representative examples. To help the reader visualize the operation of Tinker itself, an example is presented of how he or she might use Tinker to construct an alpha-beta tree search program.

2. TINKER

Tinker helps programmers visualize the operation of their programs: *Visualization* is a powerful tool in programming. Designing a program requires being able to visualize what the program should do. Debugging a program requires localizing bugs to the piece of code responsible, which is often done by visualizing the steps the program goes through and comparing the actual result to the intended behavior. One reason people find programming so difficult is that it taxes their ability to visualize procedures. The enormous amount of detail contained in successive states that programs go through overwhelms most people's ability to keep these details in their heads. Consequently, a programming environment oriented toward helping a user visualize the *operation of programs* should be very successful in making programming easier.

Tinker is an experimental system which helps a user write Lisp programs, and enables the user to "see what the program is doing" while the program is being constructed. Tinker lets the programmer put together a program step-by-step, and shows the result of each operation as it is performed. Tinker makes programming easier by explicitly displaying information about intermediate states of programs which the programmer would otherwise have to keep in his or her head.

With each piece of code in a program, Tinker associates the value which resulted from that code, to help the programmer in visualizing the effects of that code. When each operation in the program is performed, Tinker displays the output, such as text or graphics, to help the programmer visualize the progress of the program up to that point.

3. USING EXAMPLES

Tinker uses specific examples to aid visualization of programs: Programming is the art of teaching procedures to a computer. But conventional programming differs from the way in which people teach each other procedures in at least one important respect: the use of *examples*. People are much more skillful at learning procedures if a teacher presents specific examples than if the teacher presents the abstract algorithm in its most general form. Why is this so?

As each step of the algorithm is presented, the student can follow along, noting the effect of that particular step on the particular situation presented. The teacher points out which features of the situation are important and which are accidental, and the student abstracts the example to learn a procedure for the general case. When a new situation is presented, the student can check each step against his understanding of the exampie. If no example is present, the student is forced to *imagine* what the effects of each step will be on typical cases. This places a severe burden on the student's short-term memory. Examples help a student learn a procedure by giving the student a tool for visualizing the operation of the procedure. Learning procedures by examples also gives the student the opportunity to start by learning a very simple version of the procedure, then extending the procedure incrementally by considering more complex examples and special cases.

Since the power of examples in learning is so compelling, it seems strange that we should not be able to use examples in teaching a procedure to a computer. Tinker uses examples to make the programming process more natural, closer to the way in which people communicate procedures to each other. With Tinker, a program is written by presenting a specific example, and working out the steps of the procedure on that example. Tinker shows the result of each step as it is given, remembers the sequence of steps, and generalizes a program. More than one example may be shown, and Tinker has the capability to combine several examples to produce a procedure containing a conditional.

A word of caution: the reader should be careful not to confuse Tinker with previous research labeled *programming by example*. This line of research attempted to infer a procedure from the procedure's input-output history, a list of argument-value pairs. The programmer would present example inputs and desired results, without any indication of how the result should be obtained from the input. For instance, the programmer would tell the system that (REVERSE NIL) is NIL and (REVERSE '(A B C)) should result in (C B A), and the system should synthesize the usual recursive definition of REVERSE in terms of CONS. This approach met with some limited success for simple examples, but quickly becomes intractable for larger examples. Imagine showing a beginner the initial position for chess and checkmate positions, and expecting the beginner to learn chess strategy!

One problem with creating programs from input-output histories is that any given example is generalizable in a potentially infinite number of ways. The system must have some criteria for choosing which generalization to make. Any particular criteria tend to be applicable only in a limited domain, since people might want to take the same example and generalize it in different ways.

Tinker's approach hopes to retain the naturalness of presenting procedures in terms of examples, while using explicit knowledge about the procedure supplied by the programmer to make example-based programming feasible for realistic problems. Often, it is easier for the programmer to begin by working out steps of the procedure, even if he is not sure exactly what steps are necessary, than by specifying the exact form of the answer. The precise appearance of the answer often emerges only after the procedure has been observed in typical situations. Tinker's value lies in showing the programmer the results of all the intermediate steps on examples, making it much easier to detect bugs and understand the program's performance.

4. PROGRAMMING WITH TINKER

Tinker lets you write programs and debug them simultaneously: "Seeing what the program is doing" is especially important for debugging. Sometimes, of course, a program is wrong because the programmer has chosen an algorithm that is completely wrong, and the programmer must change some misconceptions and totally rewrite the program. But more often, the programmer's conception of the program is for the most part correct, but some part of the program doesn't implement what the programmer had in mind.

Finding a bug in a program is often a task of *localization* -- trying to find a specific part of the program which is malfunctioning and is responsible for the whole program's misbehavior. Localization of bugs is a matter of *examining successive states* the program goes through, and deciding at each point whether the state of the program conforms to the programmer's expectations. When a state that doesn't meet expectations is encountered, the operation which produced that state can be held responsible for the bug. Most debugging tools (such as tracing and breakpoints) are oriented towards showing the user intermediate states of the program between the start of the program, and its output.

Of course, *preventing* the introduction of bugs into a program is to be preferred to *removing* bugs once they have been introduced into a program. Tinker takes as inspiration the debugging technique of observing intermediate states of a program, and applies this technique to program construction. As a program is constructed with Tinker, the user can confirm that each step satisfies expectations. If an unwanted result is produced, the offending operation can be retracted immediately, before its effects propagate to other parts of the program. This avoids burying the erroneous operation beneath many other, possibly unrelated operations, only to have to fish it out again when some larger program of which it is a part misbehaves.

Conventional programming separates writing a program and debugging a program into two distinct activities. Since a long time passes between the time an operation is written into a program and the time the programmer discovers that the operation is the cause of a bug, it is easy to forget exactly why the operation was put there and the relationship of the operation to the rest of the program. Instead, Tinker interleaves the debugging process with the program writing process, making the introduction of bugs into programs much less likely.

5. AN ANALOGY

To illustrate the importance of displaying intermediate states in visualizing procedures, here is an analogy drawn from chess. Below are two representations of a chess game.

	White	Black
1	P-Q4	P-Q4
2	P-QB4	PXP
3	N-QB3	N-QB3

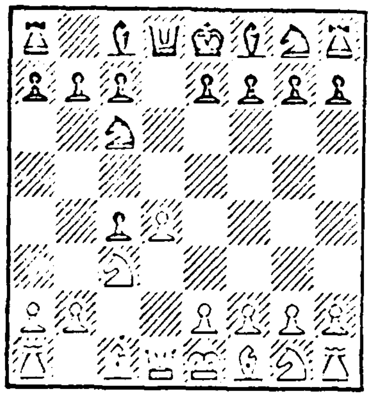


Figure 7-1

When a chess game is represented using a chessboard, it is easy to keep track of what's going on in the game. The chess player looks at the current state of the board, and uses the positions of the pieces to decide what the next move should be. The player can use the current board position to think about the consequences of each of the alternatives for the next move to be made.

When a chess game is represented only as a list of moves, it becomes so difficult to keep track of what's happening in the game that only a few, exceptional *blindfold chess* players are capable of playing in this fashion. The list of moves contains just as much information as the chessboard, yet since the intermediate states are not explicitly represented, the player must try to imagine what the board looks like after a series of moves, a staggering task for any but the most expert.

Conventional programming is a little like playing blindfold chess. When the programmer "makes a move" (writes the next function call or program statement), he must imagine what the result of that move will be on the objects he is manipulating. He must keep the current state in his head, and use the current state to decide what the next operation in the program should be. A common source of bugs is to forget or to misremember some important aspect of the current state of the program, and specify some erroneous operation.

Tinker is like a "programmer's chessboard" in that after each programming operation is specified, the result is shown immediately. Tinker's immediate, graphical feedback makes it much easier to decide what the next operation in the program should be, since it relies to a much lesser extent on the programmer's short term memory. Programming with Tinker should be easier than traditional programming in the same way that playing chess using a chessboard is easier than playing blindfold chess.

6. GRAPHICS

Examples are especially important for graphics programs: Although Tinker is independent of the subject matter of the program, the advantages of Tinker's programming methodology come through especially clearly in graphics programming. In graphics, the examples are *pictures.* The ability to "see what a program is doing" is essential for graphics programming. It is important to be able to watch pictures appear on the screen as the program is running to assess its performance. The programmer must be able to associate pieces of code with parts of the picture.

While specifications for programs which manipulate text can be given as symbolic descriptions, specifications for graphics programs are pictures. The only way to tell if a graphics program works correctly is to look at the pictures it produces and see if they look right. Thus, formal methods can never completely supplant testing for determining the correctness of graphics programs. Tinker provides an environment for constructing graphics programs where pictures appear on the screen immediately as each graphic operation is introduced into the program. The programmer can immediately see whether the operation specified produced the intended picture.

Tinker uses graphics to improve the quality of the programming environment: A goal of Tinker has been to explore how new personal computers with high resolution graphics displays can be used to radically improve the programming process. Most programming environments commonly in use today were originally designed in the days when computers were limited to character-only displays or printing terminals. With high-resolution graphics displays, the screen can be divided into *windows*, rectangular areas of the screen where text and graphics can be displayed independently. Personal computers can have pointing devices like the mouse. Our programming environments need to be restructured to take advantage of these new facilities.

In Tinker, programming happens as much as possible by selecting from a *menu*, where the system display a list of possible choices, and the user picks a choice by pointing, instead of by typing commands. This is better, especially for beginners, since the user doesn't have to remember what choices are available, or remember the syntax of commands, or be proficient at typing.

An example problem: Alpha-beta tree search: The best way to visualize the ideas behind Tinker is to watch an example of Tinker in action. Within the limitations of the paper-and-print medium, we will now try to give the reader some feel for what it is like to use Tinker for everyday programming. The problem we have chosen to present is an *alpha-beta tree search* algorithm. This is a classic problem in Artificial Intelligence, first arising in chess-playing programs. It has wide application in many problems involving two-person games, planning of actions, and problems requiring search through a space of possible situations. The program must decide what actions to take by searching a tree of possible situations. Each node of the tree represents a situation, each arc an action that can be taken to transform one situation into another. In chess, the situations are board positions, the actions chess moves.

The search proceeds by imagining the effect of each possible move and exploring its consequences. When planning an action, you say "Suppose I make this move...", then turn around and take the point of view of your opponent, imagining "Suppose he then makes this response to my move...", and planning your next response accordingly.

Situations at each node are described by a *static evaluation*, a numerical assessment of the relative advantage for the player at that node. Situations better for you are given higher numbers, those better for your opponent lower numbers. You always choose your best move and your opponent is likely to choose the action best for himself. The value of the top of the tree is determined by the maximum of the values of the nodes immediately below it. The value of the each node at the next level down is determined by maximizing the values of the nodes immediately below it, and so on, alternating minimizing and maximizing steps at each level. This is called the *minimax* search procedure.

Here is a picture of a tree of possible situations, with the leaf nodes of the tree marked with numbers indicating their static evaluations, and nonterminal nodes marked with their minimax values. We show a downward pointing arrow at a node to indicate taking the minimum of the values of branches below that node, and an upward pointing arrow to indicate taking the maximum of values below the node.

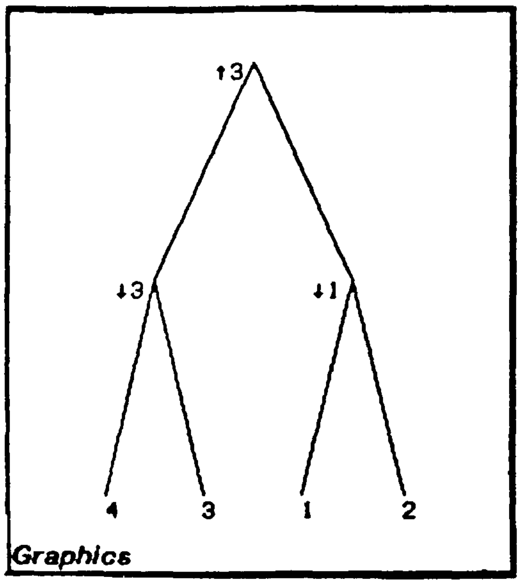


Figure 7-2

In certain situations, like the one illustrated above, it's not always necessary to explore the entire tree. The next picture shows the same tree, but captures the process of exploring the tree at a time before every node has been explored.

First we explore the left side, yielding 3, the minimum of 4 and 3. Now, imagine that we've explored the left side of the right branch, yielding 1, but have not yet explored the rightmost branch.

We can immediately conclude the value of the right side of the tree must be "at most 1", since if the number is any higher than 1, 1 would be the minimum of the two. Since the maximum of 3 and "some number which is at most 1" is 3, there's no need to explore the rightmost branch. Thus we can deduce the value of the entire tree without knowledge of every terminal node. This is called the *alpha-beta* heuristic, and it can save a lot of work in tree search problems.

By contrast, on the following tree, the alpha-beta heuristic is not applicable.

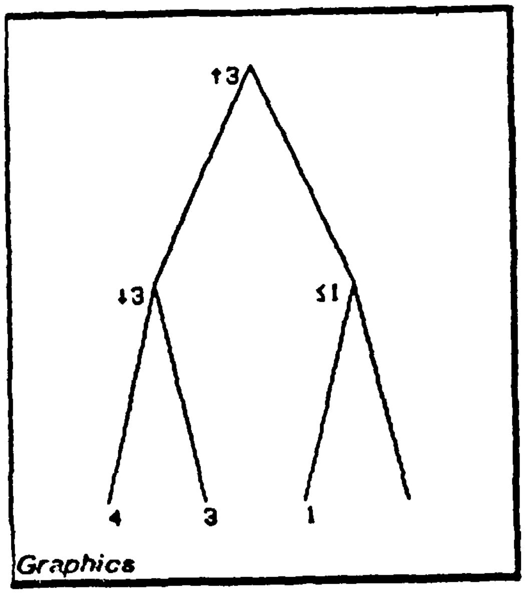


Figure 7-3

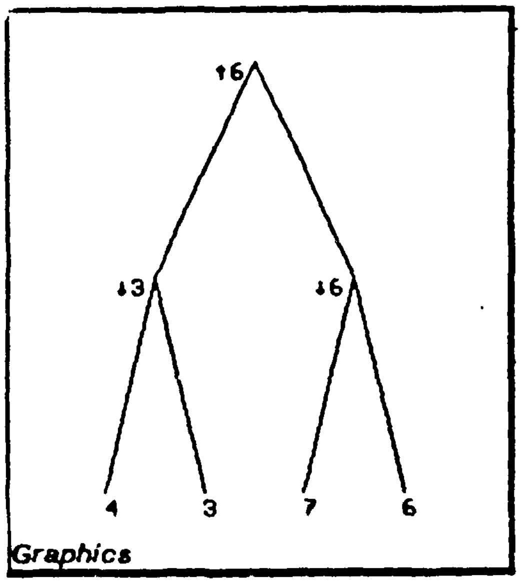


Figure 7-4

Since the value 7 of the third branch exceeds the value of the left side of the tree, 3, we are forced to explore the fourth branch. Indeed, the value of the fourth branch, 6 turns out to be the value for the entire tree in this case.

To illustrate the essential ideas clearly, we will restrict ourselves to considering a very simple variety of the alpha-beta search technique. Extensions to more complex versions, such as pruning other branches of the tree, dealing with non-binary trees, etc., can be readily imagined.

We are now going to use Tinker to develop a program to search trees using the alpha-beta heuristic. Just as the two example trees are presented to explain the alpha-beta algorithm to the reader, we will use the same two example trees to show Tinker how to perform the alpha-beta search procedure.

7. A GUIDED TOUR OF THE TINKER SCREEN

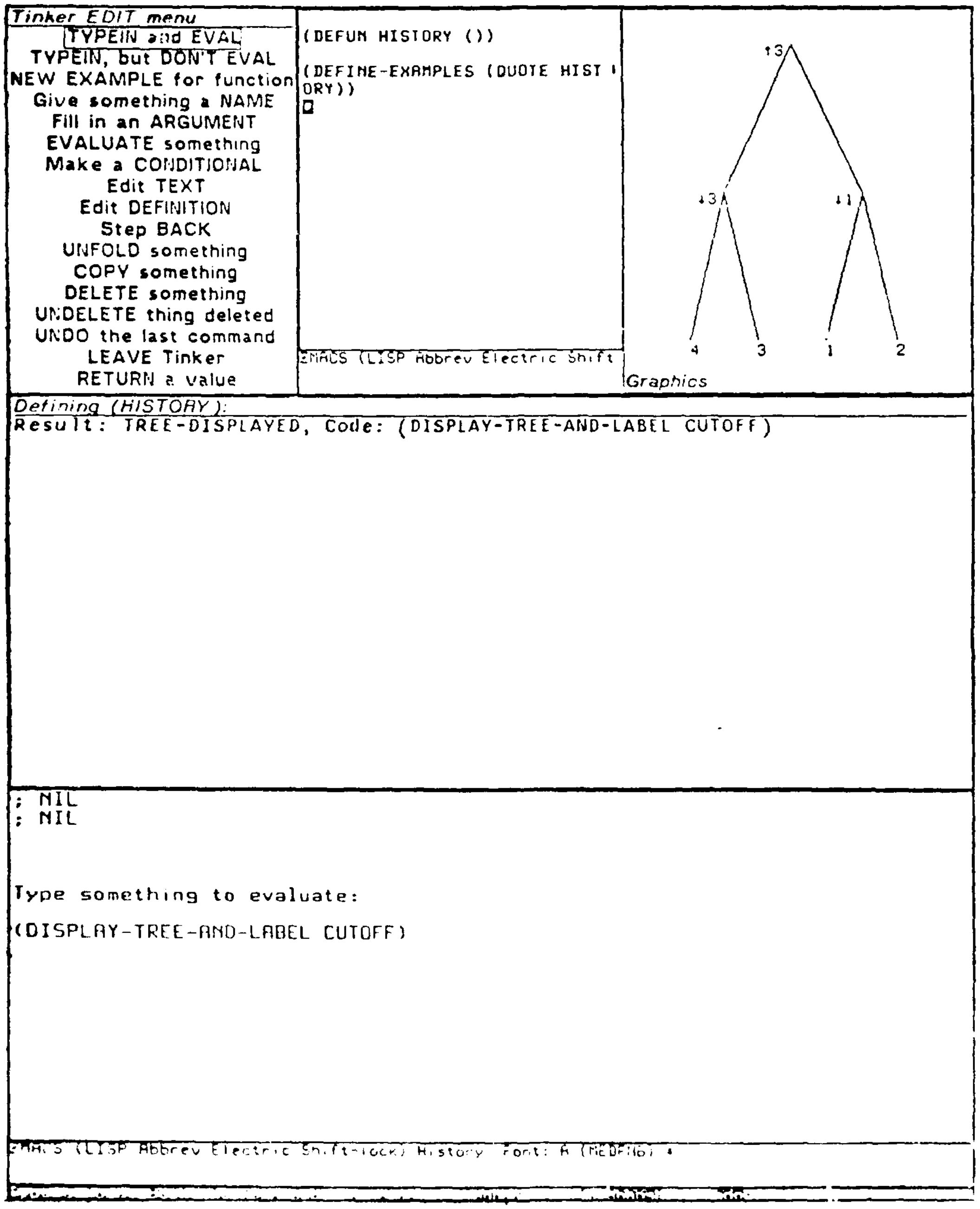
Before embarking on our project of defining the alpha-beta search procedure, we will take a few moments to explain the mechanics of writing programs with Tinker. This picture shows a typical Tinker display.

Each Tinker operation begins by choosing a menu operation from the *Edit Menu* in the upper left hand corner. In this example, we move the mouse cursor to the operation TYPEIN and EVAL, and press a button on the mouse. The TYPEIN and EVAL operation lets us enter an ordinary piece of Lisp code and have it evaluated.

Tinker then prompts us in the *Typing Window*, at the bottom of the screen, asking Type something to evaluate: and we reply by typing in some Lisp code. Whenever Tinker needs to ask the user a question or print some information, it does so in this window, and the user types all input to Tinker here. The code in this example calls an already-defined function named DISPLAY-TREE-AND-LABEL which draws trees on the screen, telling it to draw a tree stored in the variable named CUTOFF.

The title line of the *Snapshot Window* in the middle of the screen reads: Defining (HISTORY). The code which appears in the snapshot window is always considered to be code which defines the body of some Lisp function. In this case, there's a top level function named HISTORY.

As a result of the TYPEIN and EVAL operation, the text Result: TREE-DISPLAYED, Code: (DISPLAY-TREE-AND-LABEL CUTOFF) appears in the snapshot window. This displays the code entered, along with the value, TREE-DISPLAYED, produced by that piece of code. Whenever some code is evaluated to produce a value, Tinker always remembers and displays the code that was responsible for producing that value. When defining a new function, the Result: ... part of a line in the snapshot window represents the result of performing the function's steps on some particular



```
(DEFUN HISTORY ())

(DEFINE-EXAMPLES (QUOTE HIST
ORY))
```

EMACS (LISP Abbrev Electric Shift

Defining (HISTORY):
Result: TREE-DISPLAYED, Code: (DISPLAY-TREE-AND-LABEL CUTOFF)

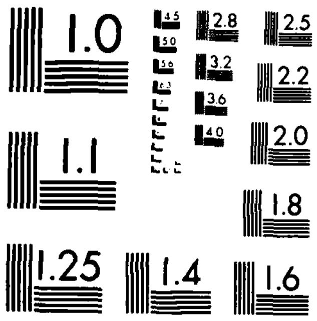
```
; NIL
; NIL


Type something to evaluate:
(DISPLAY-TREE-AND-LABEL CUTOFF)
```

EMACS (LISP Abbrev Electric Shift-lock) History Font: A (MEDFNB) *

Figure 7-5

AD-A143 459 LABORATORY FOR COMPUTER SCIENCE PROGRESS REPORT 19 1 JULY 1981-30 JUNE 19..(U) MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE.. M L DERTOUZOS 2/3

UNCLASSIFIED 01 MAY 84 MIT/LCS-PR-19 N00014-75-C-0661 F/G 9/2 NL


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

example, while the Code: ... part represents the general case for the function. In this way, Tinker can display to the user both particular examples and the code for the general case simultaneously. The commands in Tinker's command menu are mostly *editing commands* which edit the objects that appear in the snapshot window.

As a result of executing the code (DISPLAY-TREE-AND-LABEL CUTOFF), in the *Graphics Window* at the top right hand corner of the screen, we see a picture of the tree. The graphics window is used to display drawings which illustrate the behavior of the program.

The *Function Definition Window* at the top center of the screen shows the textual definition of Lisp functions generated by Tinker. Although Tinker has its own representation for programs, it produces ordinary Lisp code, which can be compiled for efficiency.

Expressions can be constructed incrementally after viewing their parts: Once Tinker evaluates some code, displaying the code and its result in the snapshot window, the programmer may use both the code and the result as part of some larger expression. The programmer can enter another function call, and specify that something displayed in the snapshot window is to be used as an argument to that function. When the function call is finally evaluated, the specific value of the argument is used to compute the value of the *function*, and the code which produced the argument becomes part of the expression for the function call. In this way, the programmer can examine the values of small pieces of code to make sure they are correct, before making them part of some larger expression.

Here's a simple example of this. We're going to display another tree on the screen, but this time we'd like to look at the printed representation of the tree *before* constructing the expression to display it.

We use the TYPEIN and EVAL operation, and type in the variable named EXPLORE-FULLY which holds the tree. The snapshot window looks like this:

```
Defining (HISTORY):
Result: #,(A TREE ((4 3) (7 6))), Code: EXPLORE-FULLY
```

This shows us the printed representation of the value of the variable EXPLORE-FULLY In this example, trees are defined to print out the numbers which label the leaf nodes of the tree. The tree EXPLORE-FULLY has a right branch whose leaves are 4 and 3, a left branch with leaves 7 and 6.

Next, we choose the operation TYPEIN, but DON'T EVAL. which puts up a piece of unevaluated code in the snapshot window.

```
Defining (HISTORY):
Result: #.(A TREE ((4 3) (7 6))), Code: EXPLORE-FULLY
Code: (DISPLAY-TREE-AND-LABEL)
```

The line in the snapshot window for the call to the function DISPLAY-TREE-AND-LABEL only has a Code: part, since we haven't evaluated it yet. Now, we choose the operation Fill in an ARGUMENT. Since DISPLAY-TREE-AND-LABEL is the only function on the screen that needs an argument, and the tree is the only thing that could possibly be the argument, Tinker immediately constructs the function call. Tinker has a policy of automatically selecting the "obvious" choice, when only one object on the screen is plausible to choose as an argument to the current menu operation.

```
Defining (HISTORY):
Code: (DISPLAY-TREE-AND-LABEL (A TREE ((4 3) (7 6))))
```

Evaluating this piece of code with the operation EVALUATE something displays the tree on the screen in the graphics window, and changes the snapshot window to:

```
Defining (HISTORY):
Result: TREE-DISPLAYED, Code: (DISPLAY-TREE-AND-LABEL
EXPLORE-FULLY)
```

Notice that the variable EXPLORE-FULLY which produced the tree becomes part of the code for the function call, rather than just the tree itself (as a constant). Whenever a value is used in further computation, Tinker carries along the code which produced that value. This shows how Tinker can build up complicated expressions one step at a time, while displaying to the programmer the result of each step.

We begin with a top-down implementation plan for alpha-beta search: When designing an algorithm, a programmer usually starts with very vague ideas about the problem, and gradually works them out to be more and more specific. In the early stages of working on a problem, it is common to have in mind some examples of how the finished program should behave, without having very definite ideas of what the code should look like. It is also typical to have a rough *implementation plan*, which maps out a strategy for implementing the task, again without committing the programmer to specific details of the code. An implementation plan might involve proposing a few major subroutines and data structures and the communication between them. Decisions made in the implementation plan are often revised in the process of working on the implementation.

In conventional programming, debugging and testing on the machine cannot proceed until a proposed solution becomes specific enough to actually start writing complete pieces of code. Tinker aims to involve the machine at an earlier stage. The programmer should be able to begin working with Tinker as soon as he or she knows some good examples for the problem, and has in mind an implementation plan which is capable of performing the procedure on the examples.

We begin working on the alpha-beta problem with a rough implementation plan. Our plan should include provision for viewing graphically the progress of the alpha-beta search as it explores the tree. Since the program is to be written by presenting specific example trees, we will be able to see dynamically what the program is doing by watching the search procedure move across the nodes of the tree.

We can do this by first displaying the whole tree by drawing only its arcs on the screen. As the search examines each node and decides on a value for that node, we will have our program label the node with its value. This will enable us to see what nodes are being looked at by the program, and in what order the nodes are examined.

Like in most programming situations, we start with a set of already-defined procedures and data structures, and these facilities are available for constructing new programs. We will assume that certain support routines and data have been defined before the start of our session, and we will not present the details of these, to avoid distracting us from the alpha-beta algorithm itself.

First, we will assume that the data structure used to represent trees has already been defined. A tree is either a LEAF node, or it has LEFT and RIGHT branches, each of which is a tree. The functions LEFT-SIDE and RIGHT-SIDE extract the two branches from the tree, and the predicate LEAF? asks whether a tree is a leaf node.

Trees may have LABELs at their nodes. We will assume that a set of example trees has been prepared for this session, including the trees CUTOFF and EXPLORE-FULLY.

We will also assume primitive graphics procedures for displaying trees on the screen. The procedure DISPLAY-TREE draws the arcs of the tree on the screen, and DISPLAY-DOTTED-TREE draws them with dotted lines. LEFT-SIDE and RIGHT-SIDE of a tree display the arcs as they traverse them. LABEL-NODE displays the label at a particular node, and DISPLAY-TREE-AND-LABEL displays a tree and labels all its nodes. We could define the tree data structure and display functions using Tinker if we wished.

We will adopt a top-down strategy for implementing the alpha-beta search. We will start with a top-level function which we will call ALPHA-BETA, which will initialize the display. This will then call a "workhorse" function AB which will compute the alpha-beta value of each node, recursively walking down the tree until leaf nodes are encountered. We will separate the work of AB into two subroutines, AB-LEFT and AB-RIGHT which compute the alpha-beta of the left and right branches of a tree, respectively. The crucial subroutine AB-PRUNE will make the decision whether or not the alpha-beta heuristic is applicable, allowing us to "prune" some branches of the tree.

The process of defining the alpha-beta search with Tinker will require three main examples. We will start by presenting the tree CUTOFF which illustrates the application of the alpha-beta heuristic. This tree will serve as the first example for the alpha-beta function. The search will be defined recursively in terms of a walk down the tree data structure until a leaf node is reached. Computing the alpha-beta value of a leaf node will be the second example, showing how the recursive procedure *bottoms out*. Next, the tree named EXPLORE-FULLY will provide a contrasting example, demonstrating that the alpha-beta heuristic is not applicable in all cases.

The first example shows how to apply the alpha-beta heuristic: We are now ready to begin writing the code for the alpha-beta search. The way we start defining a new function in Tinker is to present an *example* function call, showing a typical case in which we will use the function. We work out the steps corresponding to the procedure on the test case.

We construct a call to the ALPHA-BETA function, just as if we had already defined the function. As an example tree, we supply a tree named CUTOFF, the tree we originally used above to illustrate the alpha-beta heuristic. We use the TYPEIN, but DON'T EVAL operation.

```
Defining (HISTORY):
Code: (ALPHA-BETA CUTOFF)
```

Now, instead of evaluating that form, we instead tell Tinker that this is a NEW EXAMPLE for function, for the function ALPHA-BETA. Tinker responds by changing the snapshot window to tell us we're defining an example for ALPHA-BETA, and creates a variable to name the argument to ALPHA-BETA. We name the argument TREE using the Give something a NAME operation.

```
Defining (ALPHA-BETA (A TREE ((4 3) (1 2)))):
Result: #,(A TREE ((4 3) (1 2))), Code: TREE
```

The first action taken by the program should be to initialize the display, drawing the arcs of the tree, but without labeling any of its nodes. We use the function DISPLAY-DOTTED-TREE to display the shape of the tree on the screen, using dotted lines which will be filled in incrementally as the procedure traverses the tree.

```
Defining (ALPHA-BETA (A TREE ((4 3) (1 2)))):
Result: #,(A TREE ((4 3) (1 2))), Code: TREE
Result: TREE-DISPLAYED, Code: (DISPLAY-DOTTED-TREE
TREE)
```

In the graphics window, a picture of the example tree appears.

Now, we pass along the tree to the workhorse function AB.

```
Defining (ALPHA-BETA (A TREE ((4 3) (1 2)))):
Result: #,(A TREE ((4 3) (1 2))), Code: TREE
Result: TREE-DISPLAYED, Code: (DISPLAY-DOTTED-TREE
TREE)
Code: (AB (A TREE ((4 3) (1 2))))
```

We choose the command NEW EXAMPLE for function, which recurses, pushing

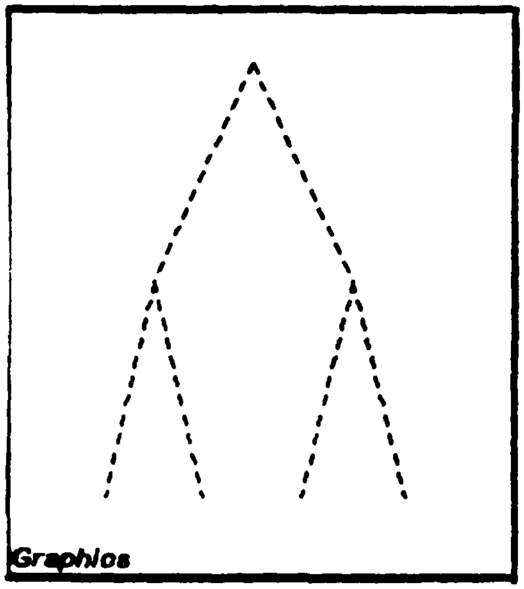


Figure 7-6

from defining the function ALPHA-BETA to defining the function AB. After we conclude the definition of AB, Tinker will return us to defining ALPHA-BETA.

```
Defining (AB (A TREE ((4 3) (1 2)))):
Result: #,(A TREE ((4 3) (1 2))), Code: TREE
```

Tinker encourages a kind of *top-down debugging*. In traditional, *bottom-up debugging*, subroutines must be defined before their callers can be tested. Tinker allows programming a top-level routine first, then when the need for a subroutine is felt, introducing an example for the subroutine.

Since we intend AB to recurse down the branches of the tree, the first action should be to extract the left branch from the tree.

```
Defining (AB (A TREE ((4 3) (1 2)))):
Result: #,(A TREE ((4 3) (1 2))), Code: TREE
Result: #,(A TREE (4 3)), Code: (LEFT-SIDE TREE)
```

We introduce a new AB-LEFT function, and provide it with the left branch of the tree as an example. We name this branch LEFT-TREE.

```
Defining (AB-LEFT (A TREE (4 3))):
Result: #,(A TREE (4 3)), Code: LEFT-TREE
```

The plan for the AB-LEFT function is to call AB recursively on each of its branches in turn, then compute the minimum value of the branches, and use that value to label the LEFT-TREE. This performs a "min" step of the "minimax" search.

First, we extract the LEFT-SIDE of the tree, since we have to recurse down two levels of the tree at a time.

```
Defining (AB-LEFT (A TREE (4 3))):
Result: #,(A TREE (4 3)), Code: LEFT-TREE
Result: #,(A LEAF (VALUE 4)), Code: (LEFT-SIDE
LEFT-TREE)
```

This yields a leaf node in our example. We recursively call AB on the left branch.

13. THE ALPHA-BETA FUNCTION BOTTOMS OUT WHEN IT ENCOUNTERS A LEAF NODE

```
Defining (AB-LEFT (A TREE (4 3))):
Result: #,(A TREE (4 3)), Code: LEFT-TREE
Code: (AB (A LEAF (VALUE 4)))
```

Taking the alpha-beta value of a leaf node is a fundamentally *different* example from computing the alpha-beta of a tree, since we want the ALPHA-BETA function to be recursive in the case of a tree, but to stop when it encounters a leaf node. So, instead of evaluating the call to AB, we tell Tinker this is a NEW EXAMPLE for the function AB.

What action should AB take when it reaches a terminal node of the tree? The AB function should just return the value associated with that node as the alpha-beta value of the node. In addition, it should display the node on the screen, using the predefined function named LABEL-NODE.

```
Defining (AB (A LEAF (VALUE 4))):
Result: #,(A LEAF (VALUE 4)), Code: TREE
Result: 4, Code: (LABEL-NODE TREE)
```

In the graphics window, the value 4 appears at the node. This demonstrates to Tinker that whenever the search procedure reaches a leaf node, it should label that node with its value, so that we can see what the search is doing. As the search progresses down the branches of the tree, it will replace the dotted lines for arcs of the tree with solid lines.

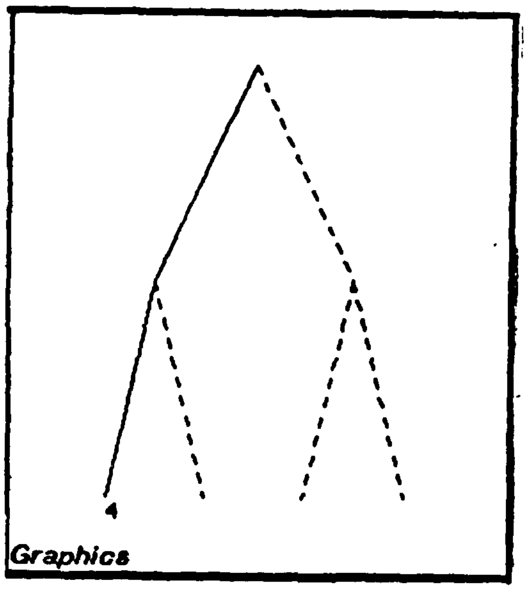


Figure 7-7

This is all we want to do to complete the leaf node example for AB, so we choose RETURN a value, returning the value 4. Tinker writes the Lisp code for AB and displays it in the *Function Definition* window.

```
(DEFUN AB (TREE)
       (LABEL-NODE TREE))
```

That definition may look silly, but it is correct for the examples we've shown it so far. Tinker develops functions by a series of *partial definitions*. As each example for a particular function is completed, Tinker produces a definition which is sufficient to make the procedure work as specified on the examples presented so far. When additional examples for an already-existing function are presented, Tinker can integrate the procedure for the old examples with the procedure for the new one. When we complete the example for AB of a full-blown tree, the code for AB will become more sophisticated. Tinker has the ability to improve the definitions of functions by adding more examples incrementally.

The search completes the left branch and proceeds to the right side:
Tinker now knows how to perform the AB of a leaf node, so we can apply the definition to the other leaf node on the left branch of the tree. This displays the value 3 on that leaf node.

```
Defining (AB-LEFT (A TREE (4 3))):
Result: #.(A TREE (4 3)), Code: LEFT-TREE
Result: 4, Code: (AB (LEFT-SIDE LEFT-TREE))
Result: 3, Code: (AB (RIGHT-SIDE LEFT-TREE))
```

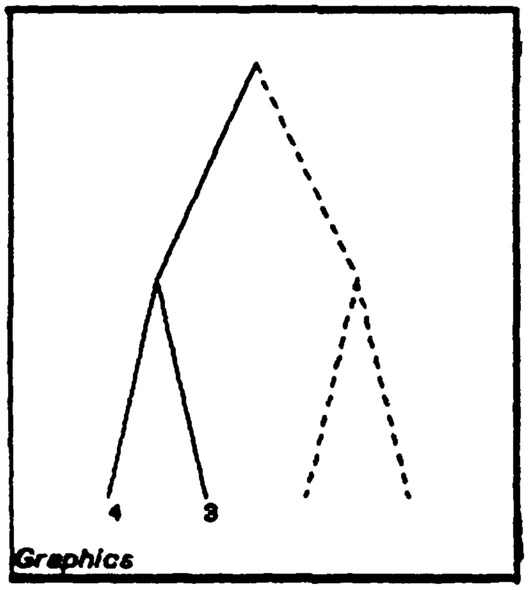


Figure 7-8

The next step is to complete AB-LEFT by using the alpha-beta values of the leaves to compute an alpha-beta value for the left side of the tree. The left branch of the tree should be labeled 3 since it should carry the minimum of the two values 4 and 3 on its branches.

```
Defining (AB-LEFT (A TREE (4 3))):
Result: #,(A TREE (4 3)), Code: LEFT-TREE
Result: 3, Code: (MIN (AB (LEFT-SIDE LEFT-TREE)) (AB
(RIGHT-SIDE LEFT-TREE)))
```

```
Defining (AB-LEFT (A TREE (4 3))):
Result: #,(A TREE (4 3)), Code: LEFT-TREE
Result: 3, Code: (LABEL-NODE LEFT-TREE "↓" (MIN (AB **)
(AB **)))
```

(The double stars "**" indicate places where Tinker elided some details of the code, since the entire code was too large to fit on one line of the screen all at once.)

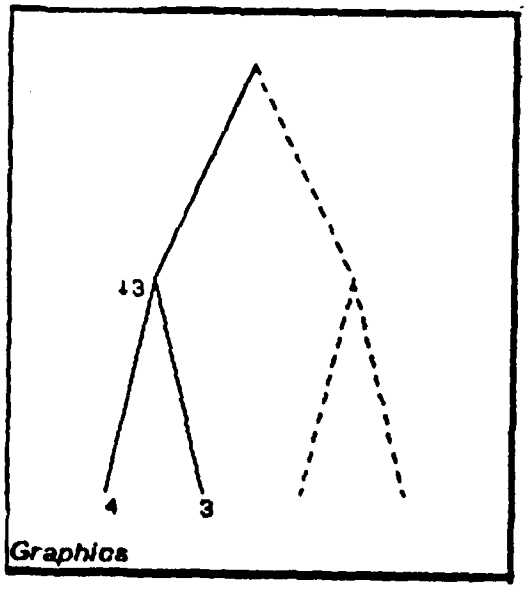


Figure 7-9

Seeing that the left side of the tree has been fully labeled, we can be assured that the definition for AB-LEFT has been completed. Tinker's ability to provide visual feedback incrementally during the construction of a program is helpful in "keeping our place" in the developing program. After choosing RETURN a value, Tinker displays the code for AB-LEFT in the function definition window.

```
(DEFUN AB-LEFT (LEFT-TREE)
       (LABEL-NODE LEFT-TREE

            "↓"
                        (MIN (AB (LEFT-SIDE LEFT-TREE))
                             (AB (RIGHT-SIDE LEFT-TREE)))))
```

After having explored the left half of the tree, the next task is to define the ction AB-RIGHT to explore the right half. If we had been doing a standard max search, the same subroutine would suffice for both sides of the tree. The s h we are going to define is asymmetrical, using the knowledge gleaned during s ng the left side of the tree to potentially save work exploring the right side.

The AB-RIGHT function needs to know the value of the left side of the tree, which we'll name LEFT-EXPLORED, as well as the right side of the tree, named RIGHT-TREE. We present a NEW EXAMPLE for AB-RIGHT.

```
Defining (AB-RIGHT 3 (A TREE (1 2))):
Result: 3, Code: LEFT-EXPLORED
Result: #,(A TREE (1 2)), Code: RIGHT-TREE
```

The third branch, the left side of RIGHT-TREE is explored unconditionally whenever we explore a RIGHT-TREE. This again makes use of the definition of AB on a leaf node that we completed earlier.

```
Defining (AB-RIGHT 7 (A TREE (1 2))):
Result: 3, Code: LEFT-EXPLORED
Result: #,(A TREE (1 2)), Code: RIGHT-TREE
Result: 1, Code: (AB (LEFT-SIDE RIGHT-TREE))
```

And now the third of four leaf nodes is marked with its value on the screen. We will

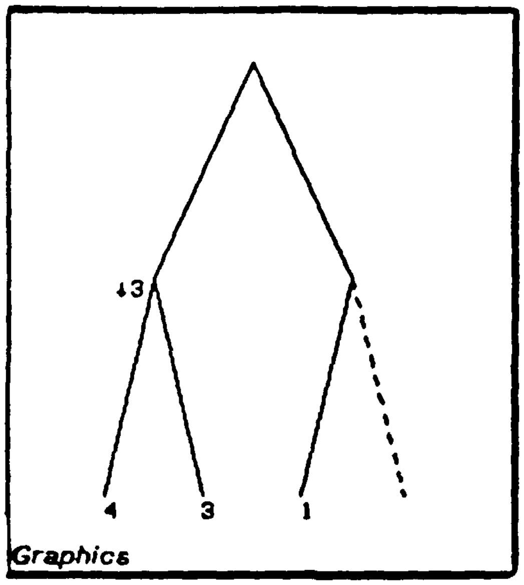


Figure 7-10

introduce the subroutine AB-PRUNE which "prunes" branches of the tree which can be ignored during the search procedure. AB-PRUNE needs the value of the third branch, which we name RIGHT-EXPLORED.

```
Defining (AB-PRUNE 3 1 (A TREE (1 2))):
Result: 3, Code: LEFT-EXPLORED
Result: 1, Code: RIGHT-EXPLORED
Result: #,(A TREE (1 2)), Code: RIGHT-TREE
```

Now, in this case, without exploring the remaining unexplored branch, we can immediately decide that RIGHT-TREE ought to be "at most 1", so we'll put a label on the tree to indicate this.

```
Defining (AB-PRUNE 3 1 (A TREE (1 2))):
Result: 3, Code: LEFT-EXPLORED
Result: 1, Code: RIGHT-EXPLORED
Result: #,(A TREE (1 2)), Code: RIGHT-TREE
Result: 1, Code: (LABEL-NODE RIGHT-TREE "≤"
RIGHT-EXPLORED)
```

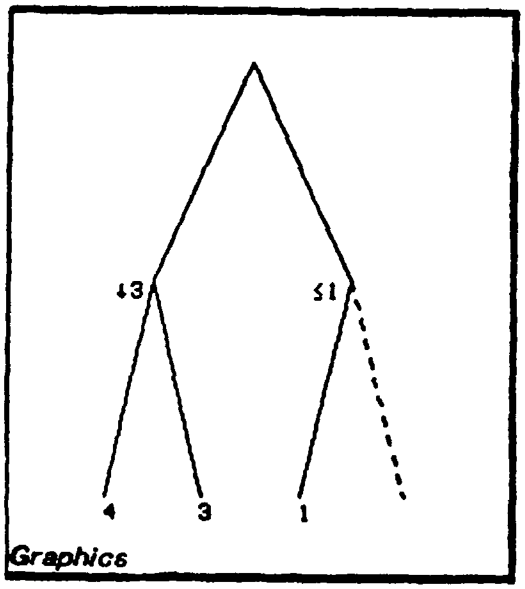


Figure 7-11

At the top level of the tree. the maximum of 3 and "at most 1" is 3 regardless of the exact value of the unexplored branch, so we can return 3 as the answer. Completing this yields definitions for AB-PRUNE and AB-RIGHT.

```
Defining (AB-PRUNE 3 1 (A TREE (1 2))):
Result: 3, Code: LEFT-EXPLORED
Result: 1, Code: RIGHT-EXPLORED
Result: #,(A TREE (1 2)), Code: RIGHT-TREE
Result: 1, Code: (LABEL-NODE RIGHT-TREE "≤"
                    RIGHT-EXPLORED)
Result: 3, Code: LEFT-EXPLORED
```

```
(DEFUN AB-PRUNE (LEFT-EXPLORED RIGHT-EXPLORED
                 RIGHT-TREE)
       (LABEL-NODE RIGHT-TREE "≤" RIGHT-EXPLORED)
       LEFT-EXPLORED)
```

```
Defining (AB-RIGHT 3 (A TREE (1 2))):
Result: 3, Code: LEFT-EXPLORED
Result: #,(A TREE (1 2)), Code: RIGHT-TREE
Result: 3, Code: (AB-PRUNE LEFT-EXPLORED (AB **) ...)
```

```
(DEFUN AB-RIGHT (LEFT-EXPLORED RIGHT-TREE)
      (AB-PRUNE LEFT-EXPLORED
                (AB (LEFT-SIDE RIGHT-TREE))
                RIGHT-TREE))
```

Returning to the definition of AB on the whole tree, we use the value returned by AB-RIGHT to label the top node.

```
Defining (AB (A TREE ((4 3) (1 2)))):
Result: #,(A TREE ((4 3) (1 2))), Code: TREE
Result: 3, Code: (LABEL-NODE TREE "↑" (AB-RIGHT (AB-LEFT
**) **))
```

We have completed the second example for the function AB, showing Tinker how to take AB of a tree, in addition to AB of a leaf node. When Tinker sees two different examples for the same function, it compares the code for the two examples. If the examples differ, Tinker asks us to define a predicate which distinguishes between the two cases. Tinker displays two snapshot windows, one showing the situation when we were defining AB on a leaf, one showing the situation defining AB on a tree. We write code that will appear *simultaneously* in both windows. The object is to define code that will yield *true* in the top window, *false* in the bottom window. This assures that our predicate correctly distinguishes between the two cases. This method of defining conditionals is especially useful in avoiding infinite loop bugs, caused by a predicate continually going down the same branch all the time.

In this case, to distinguish between a leaf node and a full tree, we write a predicate which asks the node whether or not it is a leaf.

(In Lisp, NIL represents *false*, and anything other than NIL represents *true*, so the tree in the top snapshot window answered *yes* to the question, the bottom window answered no.)

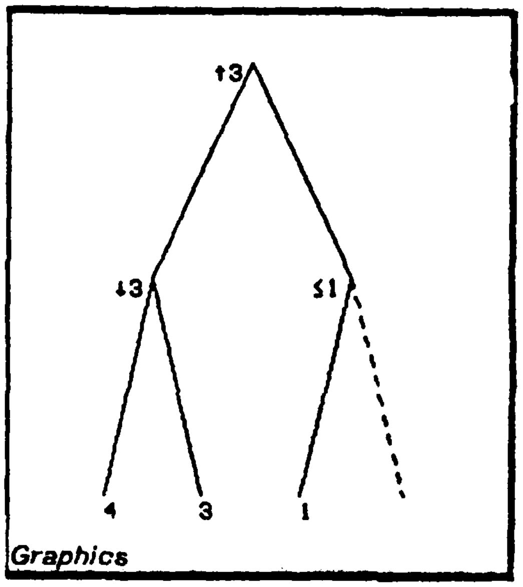


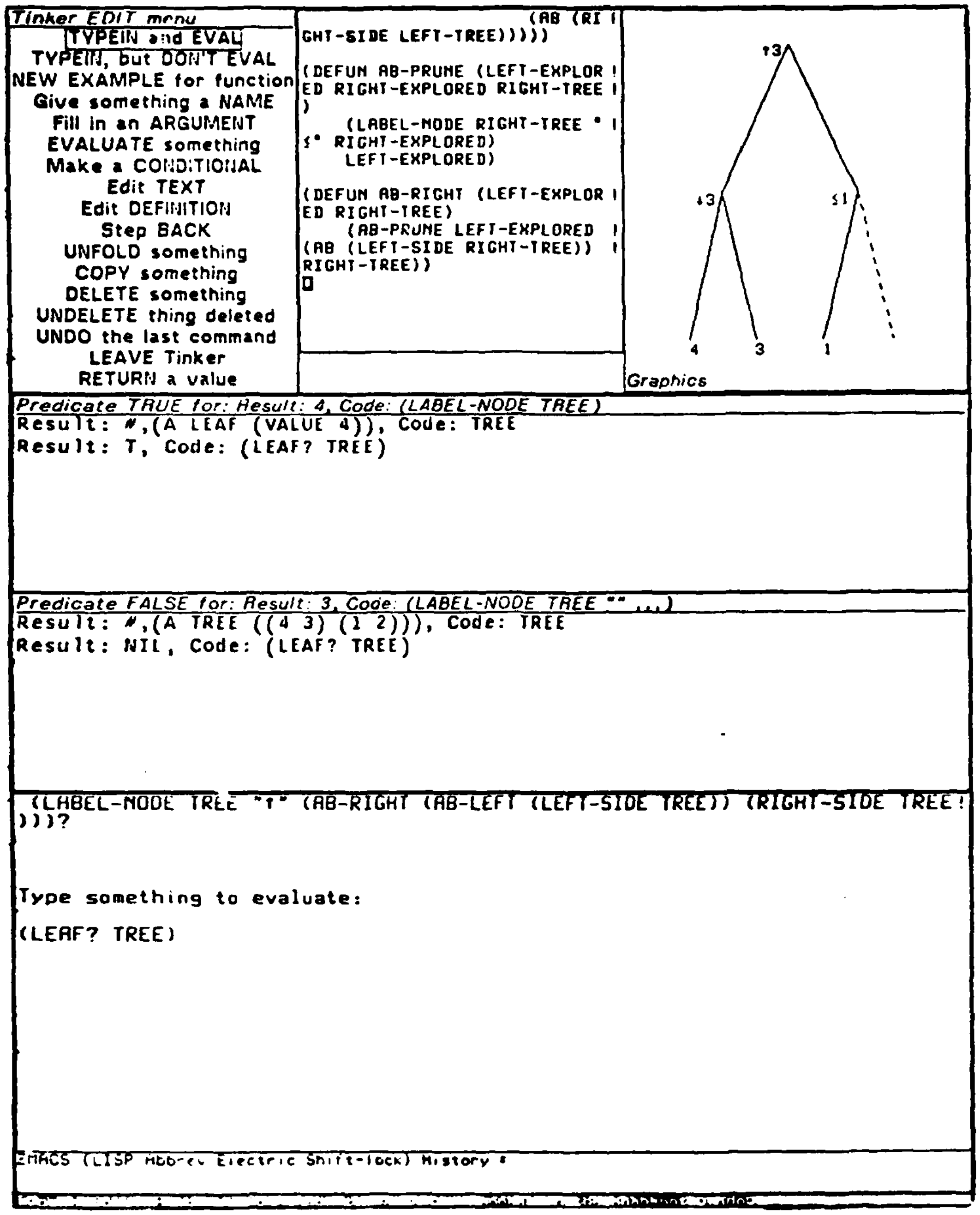
Figure 7-12

Tinker now generates a definition of the AB function containing a *conditional.*

```
(DEFUN AB (TREE)
 (IF (LEAF? TREE)
   (LABEL-NODE TREE)
   (LABEL-NODE TREE
       "↑"
              (AB-RIGHT (AB-LEFT (LEFT-SIDE TREE))
                     (RIGHT-SIDE TREE)))))
```

We could also present further examples for AB, and Tinker would create additional conditional clauses separating one case from another. For example, we should probably add to AB another case in which the argument is not any kind of a tree at all, so we can demonstrate a *negative* example as well as a positive one. The action in this case should consist of printing out some sort of error message. This is the way *type checking* can be introduced in Tinker.

This completes also the top-level ALPHA-BETA function.



```
                              (AB (RI !
GHT-SIDE LEFT-TREE)))))

(DEFUN AB-PRUNE (LEFT-EXPLOR !
ED RIGHT-EXPLORED RIGHT-TREE !
)
    (LABEL-NODE RIGHT-TREE " !
≤" RIGHT-EXPLORED)
    LEFT-EXPLORED)

(DEFUN AB-RIGHT (LEFT-EXPLOR !
ED RIGHT-TREE)
     (AB-PRUNE LEFT-EXPLORED  !
(AB (LEFT-SIDE RIGHT-TREE))  !
RIGHT-TREE))
```

```
Predicate TRUE for: Result: 4, Code: (LABEL-NODE TREE)
Result: #,(A LEAF (VALUE 4)), Code: TREE
Result: T, Code: (LEAF? TREE)
```

```
Predicate FALSE for: Result: 3, Code: (LABEL-NODE TREE "" ...)
Result: #,(A TREE ((4 3) (1 2))), Code: TREE
Result: NIL, Code: (LEAF? TREE)
```

```
 (LABEL-NODE TREE "↑" (AB-RIGHT (AB-LEFT (LEFT-SIDE TREE)) (RIGHT-SIDE TREE!
)))?


Type something to evaluate:

(LEAF? TREE)
```

```
EMACS (LISP Abbrev Electric Shift-lock) History *
```

Figure 7-13

```
(DEFUN ALPHA-BETA (TREE)
       (DISPLAY-DOTTED-TREE TREE)
       (AB TREE))
```

Another example shows the alpha-beta heuristic doesn't always work:

The program can now perform alpha-beta searches of trees -- but only for examples where we can apply the alpha-beta heuristic. At this point, Tinker has *over-generalized* the procedure to conclude that the alpha-beta heuristic works for all trees. This is not always the case for our desired search procedure.

To correct this, we can show Tinker another example. this one representing the class of trees for which it is necessary to explore the whole tree to compute an alpha-beta value. The tree EXPLORE-FULLY has that property.

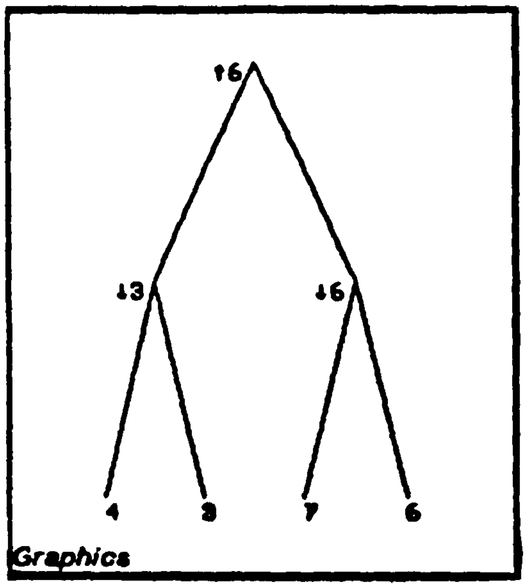


Figure 7-14

The only subroutine involved in this change is AB-PRUNE, since AB-PRUNE alone is responsible for exploring the rightmost branch of the tree. As you will recall, AB-PRUNE takes three arguments, the alpha-beta value for the left side of the tree, the value of the third branch and the as yet unexplored rightmost branch of the tree. In the case of the tree EXPLORE-FULLY, LEFT-EXPLORED is 3. RIGHT-EXPLORED is 7, and the RIGHT-TREE has leaves 7 and 6.

```
Defining (AB-PRUNE 3 7 (A TREE (7 6))):
Result: 3, Code: LEFT-EXPLORED
Result: 7, Code: RIGHT-EXPLORED
Result: #,(A TREE (7 6)), Code: RIGHT-TREE
```

We must explore the rightmost branch of the tree, and label the right tree with the minimum of the two leaves on the right side of the tree.

```
Defining (AB-PRUNE 3 7 (A TREE (7 6))):
Result: 3, Code: LEFT-EXPLORED
Result: 7, Code: RIGHT-EXPLORED
Result: #,(A TREE (7 6)), Code: RIGHT-TREE
Result: 6, Code: (AB (RIGHT-SIDE RIGHT-TREE))
```

```
Defining (AB-PRUNE 3 7 (A TREE (7 6))):
Result: 3, Code: LEFT-EXPLORED
Result: 7, Code: RIGHT-EXPLORED
Result: #,(A TREE (7 6)), Code: RIGHT-TREE
Result: 6, Code: (MIN RIGHT-EXPLORED (AB (RIGHT-SIDE
RIGHT-TREE)))
```

```
Defining (AB-PRUNE 3 7 (A TREE (7 6))):
Result: 3, Code: LEFT-EXPLORED
Result: 7, Code: RIGHT-EXPLORED
Result:#,(A TREE (7 6)), Code: RIGHT-TREE
Result: 6, Code: (LABEL-NODE
RIGHT-TREE "↓" (MIN RIGHT-EXPLORED (AB **)))
```

The value for the top of the tree is the maximum of the values for the two branches. Since the two branches of the trees have values 3 and 6, the maximum is 6.

```
Defining (AB-PRUNE 3 7 (A TREE (7 6))):
Result: 3, Code: LEFT-EXPLORED
Result: 7, Code: RIGHT-EXPLORED
Result: #,(A TREE (7 6)), Code: RIGHT-TREE
Result: 6, Code: (MAX LEFT-EXPLORED (LABEL-NODE
                  RIGHT-TREE ...))
```

This comprises a second example for the function AB-PRUNE. Tinker again creates two snapshot windows, asking us to distinguish between the two cases, one in which the alpha-beta heuristic is used, one where the tree is explored in its entirety.

The predicate which distinguishes between the two cases tests whether or not the alpha-beta value for the left side of the tree, LEFT-EXPLORED, exceeds the third of the four branches, RIGHT-EXPLORED. In both cases, the left branch evaluated to 3, but in the first case, RIGHT-EXPLORED was 1, which is smaller than 3, but in the second case it was 7, which is greater.

```
Predicate TRUE for: Result: 3, Code: (PROGN (LABEL-NODE
**) LEFT-EXPLORED)
Result: 3, Code: LEFT-EXPLORED
Result: 1, Code: RIGHT-EXPLORED
Result: #,(A TREE (1 2)), Code: RIGHT-TREE
Result: T, Code: (> LEFT-EXPLORED RIGHT-EXPLORED)
```

```
Predicate FALSE for: Result: 6, Code: (MAX
LEFT-EXPLORED **)
Result: 3, Code: LEFT-EXPLORED
Result: 7, Code: RIGHT-EXPLORED
Result: #,(A TREE (7 6)), Code: RIGHT-TREE
Result: NIL, Code: (> LEFT-EXPLORED RIGHT-EXPLORED)
```

This yields the following code for AB-PRUNE:

```
(DEFUN AB-PRUNE (LEFT-EXPLORED RIGHT-EXPLORED
RIGHT-TREE)
    (IF (> LEFT-EXPLORED RIGHT-EXPLORED)
      (THEN (LABEL-NODE RIGHT-TREE "≤" RIGHT-EXPLORED)
            LEFT-EXPLORED)
        (MAX LEFT-EXPLORED
             (LABEL-NODE RIGHT-TREE
                         "↓"
                         (MIN RIGHT-EXPLORED (AB
                  (RIGHT-SIDE RIGHT-TREE)))))))>
```

Let's try alpha-beta search on a large tree: Our alpha-beta search procedure is now complete. To illustrate its behavior, we can try it out on a large and complex example which will exercise all of the cases the program knows about. We will try it out on the following tree, called BIG-TREE. Here are successive stages of the alpha-beta program at work.

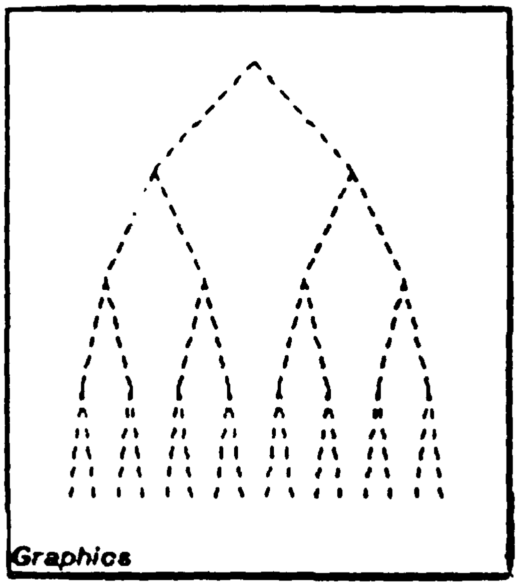


Figure 7-15

In this example, you can see two distinct alpha-beta cutoffs. The first two nodes looked at were 8 and 7, so their common ancestor is labeled with the minimum, 7. Since 5 for the next leaf node is less than 7, the program did not need to explore the next node.

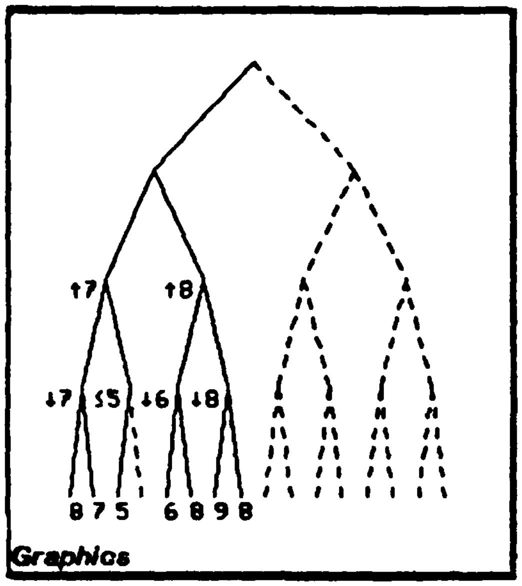


Figure 7-16

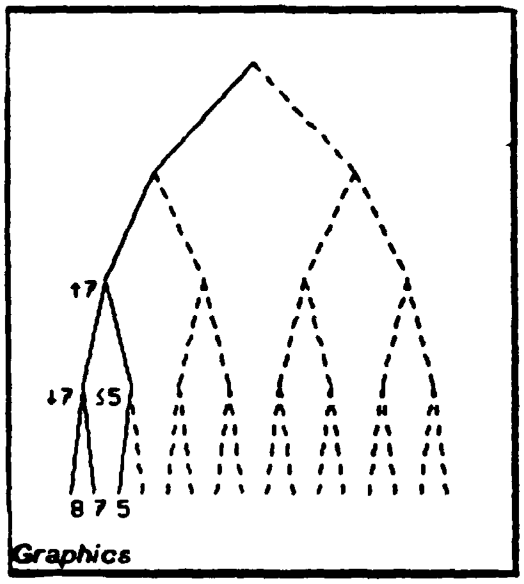


Figure 7-17

At the very top of the tree, 7 is computed for the value of the left side of the tree. The left half of the right side yields 2 which is less than 7. This time the program could cut off an entire section of the tree, rather than just the single-node cutoffs we saw previously. This saved almost a quarter of the work involved in examining the entire tree!

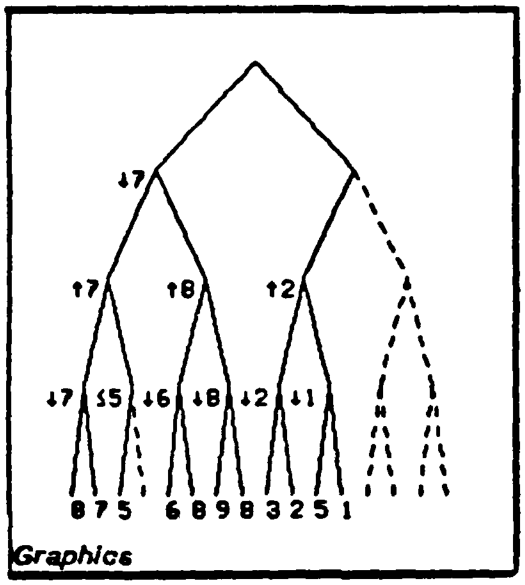
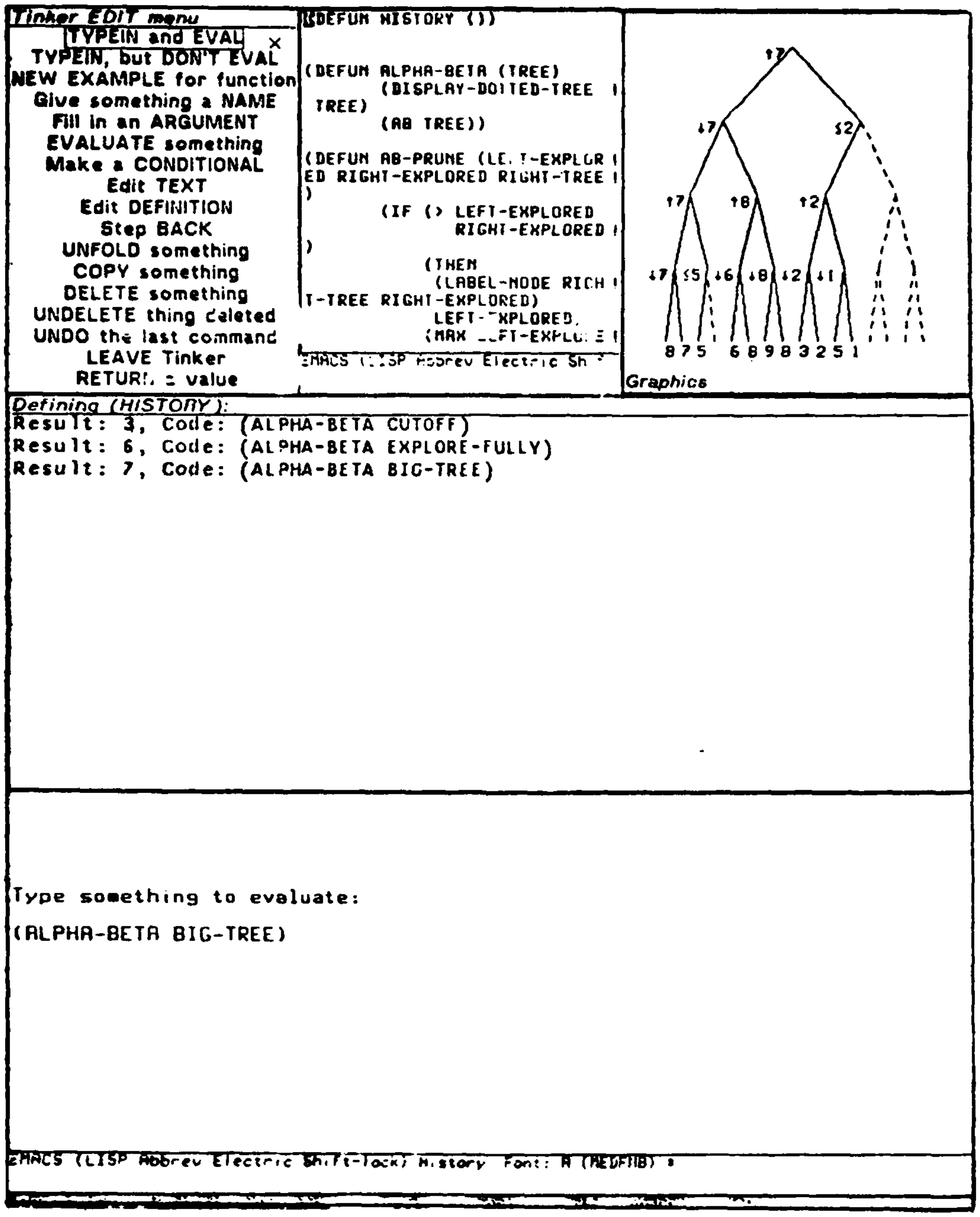


Figure 7-18

We hope this example has successfully illustrated how Tinker uses an example-based programming methodology, incremental program construction, and immediate graphical feedback to make programming easier and more reliable.



```
(DEFUN HISTORY ())


(DEFUN ALPHA-BETA (TREE)
       (DISPLAY-DOTTED-TREE !
 TREE)
       (AB TREE))

(DEFUN AB-PRUNE (LEFT-EXPLOR !
ED RIGHT-EXPLORED RIGHT-TREE !
)
       (IF (> LEFT-EXPLORED
              RIGHT-EXPLORED !
)
             (THEN
              (LABEL-NODE RICH !
T-TREE RIGHT-EXPLORED)
              LEFT-EXPLORED.
             (MAX LEFT-EXPLORE !
EMACS (LISP Abbrev Electric Sh
```

```
Defining (HISTORY):
Result: 3, Code: (ALPHA-BETA CUTOFF)
Result: 6, Code: (ALPHA-BETA EXPLORE-FULLY)
Result: 7, Code: (ALPHA-BETA BIG-TREE)
```

```
Type something to evaluate:
(ALPHA-BETA BIG-TREE)
```

```
EMACS (LISP Abbrev Electric Shift-lock) History  Font: A (MEDFNB) *
```

Figure 7-19

Publications

1. Attardi, G. and Simi, M. "The Power of Programming by Example," Workshop on Office Information Systems, Saint-Maximin, France, October 1981.

2. Barber, G. "Embedding Knowledge in a Work Station, 2nd International Workshop on Office Information Systems, Saint Maximin, France, North Holland, October 1981.

3. Barber, G., and Hewitt, C., "Foundations for Office Semantics," 2nd International Workshop on Office Information Systems, Saint Maximin, France, North Holland, October 1981.

4. Barber, G., "User Interfaces for Problem Solving Support, N.Y. University Symposium on User Interfaces, New York, N.Y., May 1982.

5. Barber, G., "Supporting Organizational Problems Solving with a Work Station, ACM Conference on Office Information Systems, June 21-23 1982, Philadelphia, PA.

6. Barber, G., "Supporting Organizational Problem Solving with a Work Station, ACM Transactions on Office Information Systems, January 1983, To appear in Proceedings of the ACM Conference on Office Information Systems, June 1982.

7. Brown, F.M. "Dynamic Program Building, "Software Practice and Experience," August 1981.

8. Byrd, R.J. Smith, S.E., and de Jong S.P., "An ACTOR based Programming System," Proceedings of the ACM-SIGOA Conference on Office Information Systems, Philadelphia, Pennsylvania, June, 1982

9. Curry, G.J. Programming by Abstract Demonstration, Ph.D. Dissertation 78-03-02,University of Washington at Seattle, 1978

10. Halbert, D. "An Example of Programming by Example," S.M thesis University of California, Berkeley, Berkeley, CA, 1981.

11. Kornfeld, W. "Applications of LISP to Music," Computer Music Journal, Vol. 4, No. 2, 1980.

12. Kornfeld, W. "A Synthesis of Language Ideas for AI Control Structures,"

MIT Artificial Intelligence Laboratory, Working Paper 201, Cambridge, MA, 1980

13. Kornfeld, W. "The Use of Parallelism to Implement a Heuristic Search," Proceedings of the 7th International Joint Conference on Artificial Intelligence, Vancouver, 1981.

14. Kornfeld, W. "Combinatorially Implosive Algorithms," Communications of the ACM, 1982.

15. Kornfeld, W., "Concepts in Parallel Problem Solving," Ph.D dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1981.

16. Kornfeld, W., and Clinger, W. "Nondeterminism and Intelligence," to appear 1982.

17. Knuth, D. and Moore, R.W. "An Analysis of Alpha-Beta Pruning," Artificial Intelligence Journal, Vol. 6, No. 4, 1975.

18. Lieberman, H. "Constructing Graphical User Interfaces by Examples," Graphics Interface '82 *Conference, Toronto, Canada*, May 1982.

19. Lieberman, H., "Testing Your Program While You Write It," Workshop on Effectiveness of Program Testing and Proving Methods, Catalina Island, California, May 1982.

20. Lieberman, H., "Seeing What Your Programs Are Doing," MIT Artificial Intelligence Memo 656, MIT Artificial Intelligence Laboratory, Cambridge, MA February 1982.

21. Lieberman, H., "Tinker: Example-Based Programming for Artificial Intelligence," Seventh International Joint Conference on Artificial Intelligence, Vancouver, B. C. Canada, August 1981

22. Lieberman, H. and Hewitt, C, "A Session with Tinker: Interleaving Program Design with Program Testing," Proceedings of the First LISP Conference, Stanford, University, August 1980.

23. Lieberman, H. and Hewitt, C. "A Real Time Garbage Collector Based on the Lifetimes of Objects," Communications of the ACM, October 1981,

24. Neves, D., "Learning Procedures from Examples," Carnegie-Mellon University, 1980

25. Pangaro, P., "The Animation System EOM," Creative Computing, November 1980.

26. Smith D.C., "Pygmalion: A Creative Programming Environment," Ph.D dissertation Stanford University, 1975.

Talks

1. Barber, G. "Embedding Knowledge in a Work Station," 2nd International Workshop on Office Information Systems, Saint Maximin, France, North Holland, October 1981.

2. Barber, G. "User Interfaces for Problem Solving Support," N.Y. University Symposium on User Interfaces, New York, N.Y., May 1982.

3. Barber, G., "Supporting Organizational Problem Solving with a Work Station," ACM Transactions on Office Information Systems, January 1983,

4. de Jong S.P., Chairperson, "Prospects for more Intelligent Office Systems," ACM-SIGOA Conference of Office Information Systems, Philadelphia, Pennsylvania, June 1982.

5. de Jong, S.P., Programming Committee, ACM-SIGOA Conference of Office Information Systems, Philadelphia, Pennsylvania, June 1982.

6. Kornfeld, W., "Everything You Always Wanted to Know About MUZACS But Were Afraid to Grovel Through the Code to Find Out," MIT Artificial Intelligence Laboratory, Cambridge, MA, January 1981.

7. Kornfeld, W., "The MUZACS Symbolic Music Editor," Le compositeur et l'ordinateur (the composer and the computer) Conference, IRCAM, Centre Pompidou, Paris, France, February 1981.

8. Kornfeld, W., "Compiling Pattern Directed invocation into Message Passing," Second Annual Workshop on Distributed Problem Solving, Dedham, MA, June 1981.

9. Kornfeld, W., "Virtual Collections of Assertions," Jet Propulsion Laboratory, Pasadena, CA. November 1981.

10. Lieberman, H., "Research at the MIT AI Lab: Act 1 and Tinker," 9 March 1982, Boston University

11. Lieberman, H., "Example-Based Programming for Artificial Intelligence," Xerox Palo Alto Research Center, Palo Alto, CA August 1981,

12. Lieberman, H., "Act 1: An Object-Oriented Parallel Message Passing Language for Artificial Intelligence," Conference on General Purpose Homogeneous Parallel Computer Architectures, Endicott House, Dedham, MA, June 1981.

13. de Jong, S.P., "Course Instructor for "Relational Data Bases", Session on Query-by-Example, Continuing Education Institute, September 1982.

OFFICE AUTOMATION

Academic Staff

I. Greif		M. Hammer
	M. Sirbu	

Research Staff

T. Anderson		R. Ilson
	L. Rosenstein	

Graduate Students

Y. Bakopoulos		M. Good
B. Berkowitz		J. Kunin
D. Carnese		B. Niamir
J. Cimral		S. Sarin
E. Gilbert		J. Sutherland
	S. Zdonik	

Undergraduate Students

A. Adamoli	A. Mondori
B. Bauman	J. Nachimson
P. Czarnecki	J. Nitchman
K. Hsu	A. Tallian
Y. Hui	S. Trieu
Y. Kim	J. Yoon
W. Mok	M. Zurko

Support Staff

C. Hengeveld

1. OVERVIEW

The activities of the Office Automation Group this year were in three areas:

- Office Studies: completion of the design of OSL, the Office Specification Language; extensions to the OAM, Office Analysis Methodology; longitudinal studies of office productivity.

- Multi-person Informational Work: design and implementation of several prototypes for cooperative calendar management activities; a facility for meeting support; design of support tools for builders of multi-person applications.

- The Office Workstation: continued design and implementation of ECOLE, the integrated multi-function workstation, including a new implementation of ETUDE, the document preparation system.

Research in each of these three areas is described in the following sections.

2. OFFICE STUDIES

Understanding and modeling office work has been a major thrust of the research in this group for several years. The Office Analysis Methodology (OAM) was developed in 1980 and tested in 1981. The Office Specification Language (OSL) was also under development during that period. In contrast to the short term descriptive analysis of an OAM studies, we have also been pursuing a program of longitudinal studies aimed at understanding impact of office automation of office productivity.

In the past year, the OSL effort has culminated with the completion of the Ph.D thesis by Jay Kunin, *Analysis and Specification of Office Procedures*. The results of the testing of OAM and OSL carried out as part of Kunin's thesis research have prompted changes and additions to OAM.

This section contains a review of the OAM research, followed by a discussion of longitudinal studies.

2.1. OAM

The Methodology as Tested: A very simple outline of OAM is given below for those who may not be familiar with it. It is followed by a discussion of the problems that have been identified and one solution that has been proposed.

OAM starts by telling the analyst how to approach the office. It discusses the reasons for doing an analysis of an office and how these different reasons can result in different approaches to the office and different problems while doing the study. For example, a study performed because upper management has said that an office is to be automated requires that the analyst work hard to achieve rapport and a helpful, friendly atmosphere with the office staff, while a study that is done at the request of the office manager and with the help of a person from the office requires less effort in these areas.

The next step in an OAM study is to meet with the office manager. This meeting has several purposes. The first is to lay a groundwork for the study. The analyst explains what OAM is and outlines the amount of time and effort that will be required from office personnel, typically 3-5 hours each. The analyst should also be certain that the exact scope and purpose of the study are defined. The next step is for the analyst to ask about the office itself: the mission, the internal structure, its place in the organization, what other offices it communicates with, and so forth. Finally, the analyst should make arrangements to have the manager tell the office staff about the study and let them know who the analyst is.

The analyst then interviews the office staff, looking for procedures, objects and functions. Each interview takes about an hour, and the analyst uses the model of the office to direct the interview. OAM provides a lot of detail about interviewing techniques. After the first few interviews, the analyst starts writing a first draft of the description. After each interview, the analyst will add to, or modify, the description according to the information obtained. When all of the interviews are finished, the analyst finishes the description, then circulates it back to the people who were interviewed for them to read. A few days later, the analyst goes back to do a second round of interviews, looking for mistakes in the description, adding material that was not obtained in the first round of interviews, and asking about exceptions and variations to procedures.

The pattern of interview, circulate description, and interview again, is repeated until the analyst has obtained a complete description of the office and all of the people who are interviewed agree that this is so. Theoretically it ought to happen in two rounds of interviews, but in practice the analyst often has to go back to at least some people a third time. When the consensus is reached, the analyst gives a copy of the final description to the manager, then goes back to interview the manager again. At this interview, the analyst asks about any changes to the description and about any general exceptions that apply to the office as a whole. Once the manager's changes and data are incorporated into the final description, the process is finished.

Problems with OAM: OAM does produce good descriptions of the current operations of offices, but there are several problems with it. First, it takes too much effort to do an OAM study, compared to the value of the results. This implies that either the amount of time required must be reduced, or the value of the analysis must be increased. In most offices, the OAM analyst must interview nearly everyone in the office at least twice. This quickly becomes very time consuming. In addition, the amount of detail that is gathered requires great effort from the analyst to organize. The result of all of this effort is simply a description of the current operations of the office. While such a description is necessary to ensure that everyone involved in the automation process starts with the same information, it is only a small part of the process. The key questions are not how the office currently works, but what is wrong with how it currently works and how can that be fixed? OAM does not answer these questions.

A second problem with OAM is that it is useful only in certain types of offices. OAM works best for offices that have at least some structure. This is not to say that it needs as much structure as most of the data processing techniques, in fact, studying an accounting or payroll operation that has not been automated will not make use of many of the capabilities of OAM. However, offices that are totally unstructured, such as offices whose sole purpose is to work on special projects, will not be easy to study with OAM and the results will be quite unsatisfactory. In addition, even within a semi-structured office, some office personnel, particularly managers, will have fairly unstructured jobs. The standard OAM interview will fail here, since there will be no structure to guide the interview and no way to find the information specified by OAM.

Changes to OAM: OAM assumes that, since at the early stage of office analysis one does not know what information will be needed later, the initial description should contain all information that might be relevant. This attention to every detail is part of the reason why doing an OAM study requires so much effort. OAM can be streamlined to some extent by reducing the amount of detail that is collected. This requires making a new assumption: that the initial study, is simply that, an initial study. This study will be somewhat different from an OAM study,and focuses on identifying problem areas. Later studies can gather additional details when it is known exactly what data should be collected. This change to OAM is a small one, and not very exciting in a theoretical sense, but it will make the new methodology much more practical to use.

Another small, but practical change to OAM, will be to incorporate the use of questionnaires. Questionnaires can help to reduce the amount of time spent in interviews by asking for much of the background data necessary to understand the context of the office. Again, there is no change to the theory here, merely current practice, but it is a change that is badly needed and will be included in the new methodology.

Extending OAM: OAM in its current form is designed to gather information about the operations of an office. It does not explicitly gather information about areas of office work that are badly in need of change, nor does it identify leverage points, those places where change will be most effective. Somehow one is supposed to be able to figure out these things from the written description of current operations. This is much more difficult than it seems. It seems that the problem is not the lack of a method for going from the description to identification of leverage points, but rather that there is information missing from the writeup that is necessary to the identification process. Furthermore, the process of identifying leverage points should be going on in parallel to the OAM study.

Those who have done OAM studies have noticed that the analyst who did the study has a very good idea as to what the problem and opportunity areas are. This information is not collected in any specific way by the methodology, rather it is something that the analyst runs into in the course of collecting the information that actually is specified.

The general structure of an OAM study--the manager interview, then two or more rounds of interviews with the office personnel, etc.--is entirely valid and will be kept in the new methodology. Suggestions about how to approach an office, the social issues, etc. remain also remain. The idea of having a mental model is useful in doing the interviews. The function and procedure model will also still be valid, although perhaps not as important as before. And in general, the idea of finding out what currently happens and describing it is valid. What will change are:

- the level of detail gathered about various processes,

- the emphasis on only describing current operations.

A New Theory: In some of the offices that we have studied, there have been cases where what seemed like problems were not, in fact, the real problems at all. For example, in the Office of Sponsored Programs, (the OSP) the officers who are responsible for the various programs were found to talk to each other a lot on the telephone. The obvious application of technology to this situation is to put in an electronic mail system. This has two results:

- the "shadow time" lost in dialing, busy signals, leaving messages, etc. would be eliminated,

- the officers could deal with questions from other officers as they have time, rather than interrupting their current activities to do so.

Further investigation in the OSP found that the officers were talking to each other trying to find acceptable contract language for sponsored research proposals they were reviewing. If a database of previous contracts and precedents appropriately arranged for easy access, were installed, a lot of the communication between officers could be eliminated. Eliminating communication always sounds dangerous, but in this case, a lot of time that the officers themselves would rather spend doing other things, would be saved. This is an example of the symptom of a problem (too much time on telephones) being absolutely clear, while the actual problem (need for particular kind of data) is obscure.

Automation can benefit an office in two ways. It can solve the problems of which the office staff are aware, and it can improve the operations of the office in ways that the staff had not expected. The problems of which the office staff are aware are usually symptoms of an underlying cause. The symptoms are usually obvious, often at the task level, but finding the real causes can be more difficult. The example above is a rather extreme one, in the sense that the cause was very deeply hidden. Most causes will be more obvious. There is, then, the following hierarchy. Office personnel will be very aware of symptoms. The analyst should be able to gather plenty of data about symptoms just by asking the office personnel what they have problems with. The office personnel may or may not be aware of causes. The analyst must always be looking for those cases where the causes are not obvious, or are not the ones that the office personnel believe. Finally, the office personnel will not be aware of opportunities. Opportunities, by their very nature, depend on some knowledge of what the available technology can do, and office personnel, with a very few exceptions, will not have this knowledge. Therefore, the analyst must identify the opportunities based on the data obtained from the study and her knowledge of the available technology.

The New Methodology: A new methodology is being developed. While the final form has not yet been developed, the current version is as follows. This methodology will be revised and tested during the next year.

1) Manager interview. This will be as described in the OAM paper, perhaps made simpler by the use of interview forms. In addition to the usual questions, however, the analyst will also ask about general problems that are known to plague the office. Answers from the manager might be things like slow turnaround of paperwork, too much paper, files cannot be found when needed, etc. These will all be symptoms which the analyst should keep in mind during the staff interviews so as to look for causes.

2) The analyst should then do the first round of interviews. The aim here is to understand the procedures, main lines and major exceptions. The level of detail should be somewhat less than was collected under the old OAM, but not too abstract either. At the end of the first round of interviews, the analyst should feel comfortable that she knows how the office works, but should not feel that she could take over any of the jobs and perform it fairly well.

In addition to understanding procedures and so forth, the analyst should ask the interviewee about any problems that he may have in accomplishing his job. This type of question, as well as questions about which tasks consume time, and how effort is allocated as compared to how it is supposed to be allocated, are all aimed at uncovering symptoms. If the analyst runs into an underlying cause it should be noted, as should hints for later follow-up, but the major emphasis in these interviews should be on finding the symptoms and understanding the general functioning of the office.

The analyst should be doing the analysis even while she is interviewing. As symptoms are uncovered, she should be looking for underlying causes in all future interviews. While, especially in the first round, the idea is not to be uncovering causes, meaningful hints may appear at any time.

3) The analyst should then prepare two items. The first is the standard OAM writeup of the procedures (excluding the database sections and the environment sections unless there is some reason to believe that they are important). The second item is a list for the analyst of all of the symptoms that have been observed, and where they fit into the procedures (if they do). We will probably make some sort of form with a page for each symptom listing what it is, which procedures it is tied to, and with spaces for working back through the why's.

4) Circulate the writeup back to the office staff for their corrections. The list of symptoms does not go back to the office staff. While the office staff are reading the writeup, the analyst should be thinking about the symptoms, to figure out what why questions might be appropriate for each, and who to ask about each one.

5) Do the second round of interviews. There are two objectives at this stage. First to be sure that the analyst has no major misconceptions about the procedures. Since we are trying to streamline the detail in the procedure descriptions, We believe that two rounds at this will really be enough and that the analyst should not have to go back again. In addition, in most cases, there should not be major changes at this point. For this reason, the analyst should call each of the interviewees and ask if they have major changes that require an interview to explain. We believe that in most cases, the respondent can write the changes onto the writeup and the analyst can call back to ask about any changes that are not obvious. This should streamline the operation somewhat.

The other objective for the second round is to uncover the causes, if any, for the symptoms. In order to do this, the analyst should have a second interview with any of the staff that she has determined to be important in answering the why's. The purpose of this interview would be to try to work backwards down the why train to the end, if possible. This may be somewhat difficult since the analyst will not be working from a model. In addition, this process may require talking to the same people several times in different interviews.

6) Prepare the final writeup and symptom/cause chart. The symptom/cause chart shows the symptoms that were found along with the cause or causes for each one and the chain that led to those causes.

7) Final manager interview. Send the materials in 6 to the manager, then get his feedback, particularly with regard to the accuracy of the analysis of the causes. The manager may know that what the analyst thought was unchangeable is in fact changeable, or vice versa. The analyst also discusses with the manager which problems can be solved with technology, which with rationalization or other organizational change, and which have no solution within the analyst's expertise. Discuss the general forms and costs of the various solutions and determine what the constraints are on feasible solutions. Explain the importance of involving the staff in any final decision and ask the manager to set up a meeting with the office staff to discuss these matters.

8) Circulate the final version of the writeup to the office staff.

9) Meet with the office staff (or representatives thereof) and the manager to discuss the symptom/cause findings. Again, the staff may suggest that the analyst was wrong in places, or may think of other items to be added. When consensus is reached about the symptom/cause list, present the possible solutions, and the applicable constraints, and get feedback from the staff concerning them.

10) Draw up a plan for future action. This can be done either at the meeting with the staff or later with just the manager. The plan should indicate what solutions will be adopted or investigated and what work needs to be done to do that.

This methodology produces three documents, the writeup describing current operations, the symptom/cause/solution chart, and the plan for future action.

2.2. Longitudinal Studies

Office systems technology is rapidly being introduced in diverse user environments. While great claims have been made for these technologies, few careful studies have been done to determine their impacts on the office. In particular, careful comparison of offices before and after the implementation of new office systems technology is required to determine the nature and extent of these impacts. During the past year we have been developing methodologies for measuring the impact of office systems through before/after or longitudinal studies and have conducted a small scale longitudinal study at MIT's Industrial Liaison Office.

The impacts of office systems technology are expected to be diverse and widespread from changes in productivity to changes in working style and relationships. Our work has focused on developing methods for measuring impacts in three areas: productivity, quality of working life, and organizational structure. Of these three, changes in productivity are the most important, and in some ways the most difficult to measure. Investments in office systems technology are justified on the basis of their contribution to office productivity; yet measuring white collar productivity has been and remains a difficult problem.

Building on our research in office analysis methodologies (OAM), we have formulated a strategy for identifying productivity measures in the office. OAM focuses at the departmental level, and attempts to identify the principal business functions of the department. Similarly, we believe it is important to look at the impacts of office systems technology at the departmental, as opposed to the individual level. We have attempted to use the functional description of the office produced in the course of an OAM study as a means of identifying the outputs which should be measured in any longitudinal study.

For evaluating the impact of office systems technology on the quality of working life we have developed a questionnaire which is based in part on the Job Diagnostic Survey developed at the Michigan Survey Research Institute.

As a test of our approach to evaluating the impacts of office automation, we have been conducting an evaluation at MIT's Industrial Liaison Office. On July 1, 1981, the ILP installed an integrated office support system based on a DEC VAX 11/780 and software support for document editing and formatting, electronic mail, a spelling checker, personal databases, and departmental databases (the latter did not become available until March, 1982).

Prior to the installation of the system, a QWL questionnaire was distributed at the ILP. In February, 1982, the questionnaire was distributed a second time and the results compared. A total of 11 persons completed both the pre- and post-implementation questionnaires. The small samples size must be considered in evaluating the significance of the findings.

The results of the comparison are complex and difficult to interpret. There was an increase in the level of satisfaction with the working environment and the resources available to perform one's tasks. Responses to the equipment itself was generally positive.

There was also an increase in the perceived pace of the work (although this may be accounted for by the fact that preliminary data were collected during the slower summer months and post data were collected in the busier month of February).

Work was also perceived to be less diverse and less challenging after the implementation of the system. On the other hand, Administrative personnel did report an increase in the opportunities to learn new skills. Support personnel felt more autonomous, and felt they had more say in how work should be done, and what resources should be allocated.

User reactions to the system appeared to be highly correlated with their prior expectations. Those who expected to like the system before it was installed generally did so. Those whose prior attitudes were more negative, generally were less happy with the final result. This suggests the importance of preparing user groups prior to the installation of new technology.

The results of the surveys will be further analyzed in future work, and a third survey will be conducted when the users have had even more time to adjust to the system.

3. MULTI-PERSON INFORMATIONAL WORK

The project on multi-person informational work is concerned with the ways that people work together on office activities and the use of computers in support of cooperative work efforts. We have developed approaches to supporting two kinds of interactions:

- In *real-time* meetings all participants are present and on-line simultaneously. The computer can supply participants with both shared and private work spaces during their meetings. The shared space is a part of each participant's screen on which all participants see the same information.

- In *meetings over time* participants work at their own pace but obey certain agreed to conventions for communication with their co-workers. The computer can support the observance of these conventions based on information about the roles and working relationships among co-workers.

Our approach has been to build prototype multi-person applications and then to analyze their implementations in order to identify and build a set of software packages and design guidelines for the builder of other multi-person application software. In the following sections we describe our prototypes for calendar management, our work on real-time communications, and requirements for database support of "working relationships." We conclude with a brief discussion of plans for the coming year.

3.1. Calendar Management and Meetings Over Time

While individual time management support in the form of personal reminders and ticklers can serve an important function, the real power of a calendar management package is in its facilities for coordination among individual calendars. If it can support scheduling of meetings among individuals (without the sacrifice of privacy of personal calendar information), access to public "resource" calendars (e.g. for conference room reservations) and notification of company-wide events from a central calendar, then an on-line calendar becomes an attractive alternative to the convenient paper calendar in the pocket. Calendar management *is a cooperative activity* and calendar management software should be designed and built with this in mind.

Roles and Working Relationships: An individual may choose to share his calendar information with others, but the calendar is his private database and he controls all access to it. Acting as the *owner* of his calendar, he may give access to his *secretary* or to certain *working group members* so that they can write in his calendar to schedule meetings. Since strangers may want to request appointments, he might also specify that members of the *public* can "write" in his calendar -- an appointment added by an individual member of the public is interpreted as a request for a meeting at a given time. A group member might only be able to cancel an appointment that he himself added to the owner's calendar while a secretary may have broader capability to modify the schedule.

A *secretary and a manager* might have to cooperate in maintaining the manager's calendar. Their working relationship definition includes

- a distinction between personal and business appointments (only business appointments would be handled by both people)
- conventions about erasures (should an appointment canceled by the secretary disappear or be brought to the manager's attention as "canceled")
- conventions about notification (how should new appointments written by one be brought to the other person's attention?)

The working arrangement might change over time and its "definition" should be modifiable.

Communications Needs: People tend to keep their own schedules in individual databases. When those databases are implemented on paper calendars, secretary/manager cooperation can be quite complicated. Often the secretary and the manager keep their own separate records of the manager's schedule and have to communicate on a regular basis to keep the copies consistent. Scheduling of meetings involves similar coordination among "group members." Just as the secretary and manager have to keep their calendars consistent, meeting participants must maintain the consistency of information about the meeting. Typically the initial scheduling is done in several passes starting with collection of scheduling constraint information followed by negotiation and selection of a meeting time. Coordination is then needed to communicate changes, e.g. to change the meeting location.

Most commercially available on-line calendars are either individual user tools or support communication by using the mail subsystem for notification. We have two comments on the use of electronic mail for scheduling meetings. First, simple notification (a one way communication) is often not adequate. Negotiation might involve a series of communications based on examination of calendar information about conflicts, etc. Second, even when simple notification is appropriate, this approach provides only a poor approximation to an intelligent tickler file. A mail subsystem will only be able to inform the user that he has new mail -- not that he has a new meeting. A calendar subsystem designed to interpret changes by users as communication might be able to report that "You have a new meeting with Joe on your calendar but it conflicts with your squash game."

Integration of messages with the calendar information becomes even more important in meeting scheduling when large numbers of messages may be received regarding a single meeting. The user should be able to choose to be notified only after a significant set of answers have been received. That may mean anything from waiting until all answers are received, to notification after a majority of people have declined to attend, to notification only if no one can attend. These kinds of summaries are generally not within the capabilities of mail facilities.

Prototypes for Calendar Management: Last year we built PCAL, a personal calendar system. It served two important purposes in the Multi-Person Informational Work project. Its code has been available for use by a large number of undergraduates who have experimented with adding a number of multi-person features. Also, last fall we made PCAL available to users of our DEC 2060. Since then, a number of people at MIT and elsewhere have been using PCAL for maintaining their own calendar information. These people have provided us with valuable feedback on our design and have been available as users and testers of some of the multi-person features.

The following versions of PCAL were implemented this year. Several were experiments with notions of roles as described above. Others explored additional kinds of coordination among calendar or modifications to the user interface of PCAL.

- *Roles -- secretary/manager/public*: This version of the calendar supports three user roles and several types of appointments. Appointments can be flagged as private or protected. Private appointments are viewable only by the manager, protected appointments are viewable by the manager and his secretary, other appointments can be viewed by anyone who reads the calendar. The secretary and manager are supported in the following working agreement: new or changed appointments entered by the secretary are brought to the manager's attention; items requiring secretary action can be found by the secretary on an "action items" list.

- *Defining Roles*: This calendar relieves the user of having to explicitly mark appointments as either private or protected by allowing him to set certain defaults according to the time at which an appointment is scheduled. It is possible to define a number of kinds of hours: business hours, office hours, private hours, etc. What is more, it is possible to name and define new roles by indicating for the new role the names of the people who can assume that role and the kinds of access they can have to each kind of hour. Thus one might define a "group member" to be someone who can read all business hour appointments, and write on the calendar to make appointments during office hours. For all other hours he may only see whether the calendar owner is busy or not.

- *A Resource Calendar*: This calendar is essentially a sign-up sheet for a conference room with two roles. A user must either act as an "allocator" or as a "requestor." Some users may be entitled to act in either role and can change roles while using the calendar. A requestor can enter appointments only if they do not conflict with appointments already in the calendar and he can cancel only his own appointments. An allocator can override a request by canceling an appointment made by a requestor.

- *A Remote "Events" Listing*: This is a calendar for public events, plus facilities in the personal calendar for integrating items from this calendar with one's own personal calendar entries.

- *Remote access to personal calendar*: Facilities for locating and reading calendars from a host machine other than the one on which the database is stored.

- *Supporting a variety of terminal types*: In PCAL, when a user is entering an appointment into the calendar, he has the option of requesting that a form appear on the screen to prompt him for information. Presentation of these forms depends on terminal type. This version of the calendar includes alternative implementations of the forms that can be used on a variety of terminals. Also, in cases where PCAL does not recognize the user's terminal type, a default terminal setting is made and the user is questioned about his terminal to see if a better quality interface can be supported.

- *"Direct Manipulation"*: Some users have found the command language interface to PCAL to be unnatural as compared to their models of paper calendars which can be written on directly. We experimented with direct manipulation of the calendar in a new feature for maintaining a "things-to-do" list as part of the calendar database. This experiment revealed limitations of the current window package which we are now addressing.

- *Pattern Matching to Retrieve Appointments*: In PCAL, appointments can be retrieved from the calendar by date and time. This version of the calendar allows a user to fill in some fields of an appointment form and then matches the partially filled form against appointments in the calendar. This provides the user with and easy to use retrieval mechanism.

3.2. Real-Time Communication

Sunil Sarin has been working on the development of tools to support real-time communication through computerized *real-time sessions* in which multiple users at their separate workstations can jointly view and manipulate a "shared information space." The participants in such a session would typically use a separate voice channel (e.g., a "conference call") to engage in discussion and negotiation. Our research is focused on the computer-based facility that provides session participants with a view of and command interface to the shared information space; we shall refer to this facility as *the system* below.

Windows: Shared and Private Workspaces: In our view, the function of a real-time session service is to provide users with a means for sharing *windows* that provide a command interface for viewing and manipulating application data. The command interface to a shared window should, as far as possible, be identical to the interface presented to a solitary user interacting with the application on his own. The ability to enter commands using this interface should be potentially available to all the participants sharing the window, unless the user in charge of the window wishes to limit this ability (e.g., by specifying that certain participants can only be "interested observers").

A reasonable means for mediating participants' attempts to enter commands should be provided. This, which we shall call *floor-mediation* by analogy with voice conferencing, may be done *manually* (by a participant designated as session *chairperson*, with aid from the system in the form of a queue of outstanding participant requests for the floor), or *automatically* by the system (which would require some policy for determining when to take the floor away from the current participant and grant the next request, e.g., if the current participant is inactive for a few seconds or has had the floor for a long period of time); a mixture of both methods, with the system mediating the floor except when the chairperson explicitly overrides it, may often be desired.

The participants in a real-time session should be allowed to create an arbitrary number of windows; a given window might be shared among the entire group, or be private to a single participant, or even be shared among some subgroup of the session participants (allowing "side conversations"). A window that is shared among two or more participants should be displayed identically on the screens of these participants. Each participant should be aware of which other participants are viewing the same window and which participant currently has the floor; the participants should be informed whenever the group structure changes (when the floor is passed, when one of the participants stops viewing the window, or when a new participant joins the group).

Each participant should be able to choose and modify the placement of shared and private windows on his workstation screen. While a given participant will enter input to only one window at a time, subject to floor mediation, he may concurrently view as many active windows on his screen as he chooses to. To avoid taxing his sensory abilities, he can choose to stop paying attention to a window (or to all of them, if he needs to respond to some outside interruption) at any time -- the other participants viewing the window should be informed thereof. The other participants should also be informed when the participant directs his attention to the window once again, at which point the returning participant should be presented with an up-to-date display of the state of the window and also with a notification of any events of significance that occurred during his temporary absence. (Such notification should also be available to a new participant when he views a shared window for the first time.)

Participants should not only be able to create multiple windows, they should also be allowed to define relationships among them (such as to see different "views" of the same information, based on access rights or levels of detail, in different windows), and to perform commands that span window boundaries (such as copy information from a shared window into a private window, edit it privately in that window, and write the revised information back into the shared window when ready).

In our view, the function of a real-time session service is to provide users with a means for sharing *windows* that provide a command interface for viewing and manipulating application data. The command interface to a shared window should, as far as possible, be identical to the interface presented to a solitary user interacting with the application on his own. The ability to enter commands using this interface should be potentially available to all the participants sharing the window, unless the user in charge of the window wishes to limit this ability (e.g., by specifying that certain participants can only be "interested observers").

A reasonable means for mediating participants' attempts to enter commands should be provided. This, which we shall call *floor-mediation* by analogy with voice conferencing, may be done *manually* (by a participant designated as session *chairperson,* with aid from the system in the form of a queue of outstanding participant requests for the floor), or *automatically* by the system (which would require some policy for determining when to take the floor away from the current participant and grant the next request, e.g., if the current participant is inactive for a few seconds or has had the floor for a long period of time); a mixture of both methods, with the system mediating the floor except when the chairperson explicitly overrides it, may often be desired.

The participants in a real-time session should be allowed to create an arbitrary number of windows; a given window might be shared among the entire group, or be private to a single participant, or even be shared among some subgroup of the session participants (allowing "side conversations"). A window that is shared among two or more participants should be displayed identically on the screens of these participants. Each participant should be aware of which other participants are viewing the same window and which participant currently has the floor; the participants should be informed whenever the group structure changes (when the floor is passed, when one of the participants stops viewing the window, or when a new participant joins the group).

Each participant should be able to choose and modify the placement of shared and private windows on his workstation screen. While a given participant will enter input to only one window at a time, subject to floor mediation, he may concurrently view as many active windows on his screen as he chooses to. To avoid taxing his sensory abilities, he can choose to stop paying attention to a window (or to all of them, if he needs to respond to some outside interruption) at any time -- the other participants viewing the window should be informed thereof. The other participants should also be informed when the participant directs his attention to the window once again, at which point the returning participant should be presented with an up-to-date display of the state of the window and also with a notification of any events of significance that occurred during his temporary absence. (Such notification should also be available to a new participant when he views a shared window for the first time.)

Participants should not only be able to create multiple windows, they should also be allowed to define relationships among them (such as to see different "views" of the same information, based on access rights or levels of detail, in different windows), and to perform commands that span window boundaries (such as copy information from a shared window into a private window, edit it privately in that window, and write the revised information back into the shared window when ready).

RTCAL: The Real-Time Calendar: As an example of real-time communication, we have completed the implementation of a first prototype system, RTCAL, that allows users to exchange calendar information in real time in order to select a mutually agreeable time for a meeting. The "shared window" in this system displays the aggregation of the participants' schedule cards, indicating which blocks of time are free and which are unavailable because one participant or more has a conflict.

Commands are provided to explore this shared space to find suitable meeting times, and to call a vote by "proposing" a specific time. Only one participant at a time can enter these commands, and the permission to do so (*control* of the floor) is mediated by the session *chairperson*. A special set of *control commands* is provided to allow participants to request control (such requests being queued until granted) and for the participant currently in control to give it up; the chairperson can also at any time *preempt* control of the shared space.

A participant can temporarily leave the session at any time, and when he returns he will get an up-to-date display of the shared space, together with a list of meeting times proposed in his absence. A section of each participant's screen is reserved for a "private window" in which detailed information from the participant's calendar, not shown in the shared space, is displayed in order to aid him in decision making. The private window is always kept current with respect to the shared space by updating it whenever the latter is moved to show a different date and time.

This initial experiment in real-time communication has provided considerable insight into the issues that arise when trying to support such collective activity on-line. The user interface turned out to be more complicated than anticipated. The design of good interfaces for the solitary user is hard enough and is the subject of much current research; the situation in a real-time session is exacerbated because many things can be changing simultaneously and rapidly, not under the control of the participant observing the session from his workstation. With both session status information and application information on the screen, it is hard for a participant to notice changes to one while concentrating on the other.[4] In our experiment with

[4]Some of these problems are exacerbated by the limitation of existing alphanumeric display terminals, the interface being particularly bad on terminals that do not support any form of highlighting, and would probably be alleviated with the larger screens and color capabilities of more modern workstations.

RTCAL, we have developed session implementation techniques of general applicability, which we will be using in future prototypes. An example is the *session initiation protocol*:

- *Current protocol:* Available participants are looked up in a "catalog" or "name server" that specifies where session invitation messages should be sent. Both the invitation and the reply carry an identifier for a "port" or "socket" to which session-related communications are to be directed. This message exchange can also carry application - specific information needed to properly set up the session. In the case of RTCAL, the invitation from the chairperson carries a description of the meeting to be discussed and scheduled, while the reply from a participant carries information from that participant's calendar for presentation in the shared space.

- *Future protocol*: we are examining additional methods for making initial rendezvous, such as posting the session invitation in a predetermined shared file in order to allow any user who notices it to join the session, without changing the basic invitation-reply sequence.

Prototype Designs: We are applying our experience with RTCAL to new prototype systems. Two of these, a *general-purpose session controller* for sharing existing interactive programs, and a *joint document editing system*, have been designed and are in the initial stages of implementation. We are also designing a next-generation meeting scheduling system that will provide a more flexible and powerful interface than RTCAL, and will also provide better integration of real-time sessions with "delayed" communication services (the latter not being supported in RTCAL).

The *general-purpose session controller* can be used by a group of users to conduct a joint interaction with any existing interactive application program. The controller views the program to be shared as a mapping from an "input stream" to an "output stream." These streams are usually connected directly to the individual user's terminal, but can in fact, be directed to a "pseudo-terminal" that can be controlled by another program. The session controller takes this approach and allows input to the program to come from any of the participants in the session, subject to a floor-mediation policy such as the ones described above, and to broadcast program output to all participants' terminals.

The general purpose controller will facilitate the following:

- Support for participants with *different terminal types.* The shared program will be made to believe it is interacting with some standard terminal type having only those capabilities (such as cursor motion or insertion and deletion of lines and characters) that every participant's terminal can support. Output from the shared program will usually contain terminal-dependent "escape sequences" for performing screen operations; these can be translated by the session control program into sequences that achieve the desired effect on each participant's terminal.

- Support for *multiple windows* viewing different programs. Users will be allowed to specify that a given shared program use only a certain section of the screen for output. The session control program will define the shared program's terminal to have the desired size and will translate operations on this "virtual screen" into operations on the corresponding section of participant's physical terminal screens. The session control program will also remember the state of each shared program's virtual screen, allowing users to overlay windows without losing the state of any.

- Support *temporary absences* of participants from a session. A copy of the shared program's screen state will be supplied to a participant returning from a temporary interruption; the same can also be done for a new participant joining the session late. Alternatively, a returning or newly-joined participant could be given the option of "replaying" a transcript of program output -- this would allow him to review the activity that took place in his absence, although it would take him longer to "catch up" with the others in the session.

The advantage of the above general-purpose session controller is that any existing interactive program can be easily shared in a real-time session without modification; session participants would interact with programs that are already familiar to them. However, we view it as a short-term tool, whose usefulness will be limited in future computing environments that consist of distributed high-performance workstations. The main limitation of the general-purpose session controller derives from its very generality -- it makes few assumptions about the application program being shared in a real-time session, other than that the program responds to characters typed on an input stream and produces characters on an output stream. Since the application program is viewed as a "black box," its internal functions are not visible to the session controller. It is thus impossible to separate various application modules (such as keystroke parsing, command execution, display, and file system or database interaction) and distribute some of the data and computation among session participants' workstations. There are a number of other difficulties having to do with protection and identification of users.

An alternative approach to real-time sessions is to integrate rather than separate session control and application functions. This approach is exemplified by RTCAL, described above, and by the *joint document editing system*. This system will allow a group of users to jointly view and edit text documents, using the command interface of an existing display-based editor. The system will support simultaneous editing in multiple windows, shared and private. Unlike the general-purpose session controller, however, the joint document editor will provide commands for moving text between windows, be they shared or private.

Plans: We are also studying the integration of real-time sessions with delayed communication services, with the objective of presenting users with a uniform interface regardless of which "mode" of communication they may be using at a given time. As we have noted, it is our premise that communications about application information, should be an integral part of the application database and should not require separate general-purpose mechanisms such as electronic mail. The same mechanisms used by people communicating asynchronously over time (such as object histories, structured annotations, and voting on proposed changes) should be available without modification to the participants in a real-time session.

One mechanism we believe will be useful in this context is that of placing a "watch" on application objects of interest to a user; its purpose will be to provide the user with a notification whenever some other user modifies the watched object (or performs any of some specified set of operations on the object). This will be a useful tool not only for long-term tracking of changes to an object, but also in real-time communication. A group of users viewing a shared object through one window would be informed if the same object were updated via another window. The participants would be able to specify whether the window should be automatically updated when such a change occurred. Alternatively, they might prefer to see a "data changed" message that persists until such time as they explicitly request to see the updated information.

This same mechanism should also provide a basis for users to rendezvous and initiate real-time sessions. Thus, two users who discover that they are concurrently working on the same document would be able to start a joint editing session if they so desired. Such a facility would permit users to conduct many impromptu real-time sessions that they would not otherwise have held because of the need for prior coordination; this would make a real-time session service all the more useful.

We will be exploring the above issues in a prototype system that supports meeting scheduling through both real-time and delayed communication. While we are at present building separate prototypes for specific applications (such as meeting scheduling and document editing), we are using similar protocols and data structures in the construction of these systems. As we gain experience with these techniques, many of which we have presented here, we expect to be able to develop a collection of "software tools" that can be coupled with application modules in order to realize the desired real-time session interface.

3.3. Workstation Database: Roles and Working Relationships

We envision the office workstation as presenting alternative views to the workstation database. The following sections suggest database facilities and data structures that could be used to support multi-person applications on the office workstation.

Database Organization for Calendar Management: For calendar management we suggest that the calendar be organized so that

- users assume a *role* when using the calendar (such as owner, secretary, public)

- items in the calendar are annotated with information about roles

- items in the calendar are annotated with information about status of the item:who wrote it, whether it is "new" from the point of view of certain users,etc.

- descriptions of working relationships are maintained in the database.

The calendar management subsystem would then be able to support cooperation in the following ways:

- the calendar information can be presented in a manner appropriate to one's role (e.g. only business appointments will be displayed to the secretary).

- users can be notified of *new* items, *changed* items, etc.

- users can be relieved of composing text for messages to be sent to, e.g., meeting participants, whenever all of the necessary information can be gathered from calendar entries.

Notification can take many forms from highlighting new appointments when they happen to be displayed, to flashing a warning on the screen that there are new appointments to sending mail to participants who are unlikely to look at their calendars. Whether or not there is notification can be role or working relationship dependent -- the manager might want to be informed of new appointments that are early in the morning, but might prefer not to have special notification of appointments during normal working hours for which he regularly checks his calendar. This agreement can be described in the working relationship part of his database.

Communication about meeting scheduling can be accomplished in a similar manner. A meeting can be highlighted on a participant's calendar when it is a request requiring his answer. It may remain flagged as tentative while its time is being negotiated. It may become highlighted on the caller's calendar once all participants have responded to the requests. A summary of the caller's calendar might include the notice that "All the answers are in on another meeting." The caller can then examine a report on the answers and decide either to schedule the meeting or to continue negotiations.

Another Example: Word processing subsystems of workstations provide support for the individual worker and do not directly support groups of people who must cooperate in the work of producing a document. In a multi-person document writing system, an *author* may be someone who can create, add to or modify a document. An *editor* can change an existing document. A *commentator* might be able to use "blue-pencil" to edit a document. That is, he can indicate need for changes but old and new versions are visible and can be reviewed by the author. Publication consists of making the document available to the public in some form. A *publisher* may have this ability to "release" a document. *Co-authors* of a document may cooperate in a number of possible working relationships. If they work as equals, each may have authority to make final changes to the document, although in practice co-authors may want to review each other's changes. Co-authors need not have equal status. Each may have primary responsibility for a section of the document (one author may act as editor of his section and commentator on others). When authors have different skills and knowledge, as with an *engineer* and *technical writer*, many passes of editing, commenting, and rewriting may be required to meet the standards of both.

Co-authors need to be able to inform each other of the status of their work, to request comments from other authors, and to write comments on each other's work. When working on documents written on paper, group members often use the document itself as the communication medium. Marks in blue-pencil and comments on margins allow editors and commentators to communicate with authors. If an author gives copies of his paper to a number of people, he may compare individual's comments or even collate their comments onto one master copy before proceeding to modify his text.

When working on-line one might choose to simulate the paper approach by keeping separate copies for each author, editor, etc. However, one should not underestimate the value of that blue-pencil when shifting from paper to on-line word processing. Collating of comments and coordination of divergent copies can be very difficult on-line since it so easy for individuals to make widespread changes that completely overwrite old copy. Comments may be mixed in with the text and authors can easily be confused as to which version of the documents (their own or one that an author revised) they are viewing.

To support joint document writing on-line we suggest that documents be stored in a database that is organized as follows:

- the documents should be structured: chapters, sections, paragraphs, sentences should be separate items in the database

- multiple versions of important components of each document should be maintained (such as versions reflecting changes of different individuals)

- comments should be components of documents that can be associated with other components (so that a comment can be "about" a particular sentence)

- information should be maintained for each document about roles, working relationships and actions by individuals.

With this data a joint document writing subsystem can provide user friendly interface to "blue-pencilled" text with the following options:

- a comparison of old and new versions

- old versions with comments in a side margin

- the new version with changed sentences highlighted

- a summary of the "status" of the documents such as an outline in which a chapter heading is highlighted if that chapter has been changed.

As in the calendar, these presentations can all be role specific. Thus changes could be highlighted only if they are relevant to the user *in his current role*. For example, one might see different changes highlighted when one acts as "author in charge of chapters one and two" than one would as, say "publisher." Notification can be done automatically, where appropriate, so that an author might be informed that, "Your co-author has completed his editing changes to section three."

3.4. Plans

Our immediate plans for further research on cooperative office work are in the following two areas:

- user interface design -- to determine appropriate presentation of information, for example, to aid users in distinguishing their own work from that of others

- database organization -- to abstract from our prototype implementation general principles for combining working relationship information with application specific information.

Spanning both of these areas is research on the user interface to the database: we will develop an end-user language for defining new working relationships. The definitions will be in terms that are relevant to the working situation, but will be translated into definitions of database structures and constraints on views of the data.

Longer term goals for the project include extension of our approach to a wider range of cooperative activities and implementation of cooperative applications subsystems on an integrated multi-function workstation. We are beginning implementation of a prototype joint document writing system and are developing scenarios for additional applications. Future implementations of the calendar will use the database facility currently being impiemented as part of ECOLE, described in the next section.

4. THE OFFICE WORKSTATION

Our research in office workstations during the past year has been in three areas: providing a general operating environment for a workstation, reimplementing the ETUDE document system, and designing other systems that will run on the workstation.

All systems running on the workstation operate under a general operating environment called ECOLE, and use the ECOLE command handler (parser) and window (screen) management system.

ECOLE is written in the programming language MDL. A machine independent version of MDL is being developed by another group at the M.I.T. Laboratory for Computer Science. This system, known as MIM (Machine Independent MDL) will allow us to transfer our system to other computers with a minimum of effort. We are cooperating with the group developing MIM, and helping their development, so that we may use the MIM system as soon as possible.

We are also extending the MDL language, adding support to its TYPE mechanism, to allow us to manipulate objects in the system independently of knowing their actual representation. This will allow us to represent objects manipulated by the workstation differently, without making any changes to the software that operates on those objects.

We first describe ETUDE and its subsystems: these include a spelling checker and a table system as well as a prototype page layout subsystem. Following that are discussions of the various ECOLE components, including the command handler, the window management system, and the database. Finally, the extensions to MDL are described and requirements for a graphics facility are presented.

4.1. The ETUDE System

We are completely reimplementing the ETUDE document system. In the original system, ETUDE handled its own command input and display output. In the current system, we are interfacing instead to similar routines in ECOLE. More fundamentally, however, we have redesigned the underlying document representation for several reasons:

- The original document representation was inefficient in its use of storage space, and this had a negative effect on the speed of the system. Our new implementation uses storage space more efficiently, and will be more responsive to user commands.

- In the old document representation, the document's editorial structure and outward appearance were inextricably intertwined. In the new implementation, the outward appearance is completely separated from the editorial structure, allowing the same document content to appear differently.

- The old document structure only allowed text characters to be inserted in a document. The new document structure is more flexible and allows different objects to be included in a document.

- We switched programming environments (from CLU to MDL) to gain flexibility in

- bringing our system up on various hardware platforms, such as the Apollo or the VAX.

In the new version of ETUDE we are correcting a deficiency in the user interface of the previous version. The original implementation of ETUDE required the user to be aware of, understand, and maintain the hierarchical document structure. Even when the user did understand the hierarchy, the system was sometimes cumbersome to use, and mistakes were easily made. For example, to insert a section in the document, the user had to position his cursor between two existing sections. Not only was this difficult to do, it was also hard to judge if the cursor were correctly positioned; the cursor could appear at the same location on the screen even though it was actually in different positions in the editorial hierarchy. If the user's cursor was positioned inside a paragraph when he created a section. the section would be created within the paragraph, resulting in an incorrect document structure.

The new implementation of ETUDE will enforce a document structure (or document grammar). This grammar will indicate, for example, that a section must be contained directly within a chapter in the document. With this notion in the system, the user need not be aware of the editorial structure hierarchy. When he wishes to insert a section, all he does is issue that command, and the system will determine, from the grammar, the appropriate place to actually create the new section.

In addition to simplifying the user interface, we are making other changes to decrease the time it takes to respond to a user's command. In particular, we have developed some new schemes to increase the system's response time when the user is doing simple editing, such as inserting and deleting characters.

4.2. The Spelling Checker

The spelling checker is an interactive spelling checker and corrector, similar in many ways to the DEC-20 SPELL program and the ITS SPELL program.

The spelling checker has been implemented in the MDL programming language on a DECSystem-20. It is intended to run as a subsystem under ETUDE (referred to here as the spell subsystem). Because the checker is not yet fully integrated with ECOLE and ETUDE, the user is always in the spell "subsystem." The commands currently available for the spelling checker are given CHECK and CORRECT. When integration is complete, it is expected that the user will ask to CHECK or CORRECT his document while he is editing it in ETUDE.

The user interface and document interface are, for the most part, provided by ECOLE and ETUDE and simply used by the checker. The current version of the checker checks text files rather than ETUDE documents, and this current version will be described. In addition, the changes to be made for checking an ETUDE document will be noted. A description of a typical checking scenario follows.

The user selects a command or asks for help. If the user asks for help, the list of commands and their descriptions are displayed. After a CHECK or CORRECT command, ECOLE passes control to the checker.

The algorithm used by the checker is similar to that of other interactive checkers. First, the user is asked if he wants to load a dictionary. If so, the checker loads the user's dictionary before it begins checking. It displays "CHECKING" on the screen so the user will know that it is working. The checker proceeds through the text file one word at a time, looking each word up in the dictionary. When a word isn't found in the dictionary, it is displayed in capital letters followed by a portion of its context as it appeared in the text file. If an ETUDE document is being checked, the word will be highlighted on the screen. The text of the document will already be displayed so there is no special handling of the word's context.

Next, the checker asks the parser to prompt the user with a question mark and get his command. The table below lists the user's possible responses when a misspelled word is found. For some responses, the checker calls the parser again to prompt the user for further information (e.g., for a filename if the command is "load" or "save," or for a word if the command is "replace").

COMMAND	MEANING
ACCEPT	Accept the current word
INSERT	Insert word into personal dictionary
GUESS	Display a list of guesses
REPLACE	Replace current word with typed word
CLEAR	Clear personal dictionary
LOAD	Load a personal dictionary
SAVE	Save personal dictionary
+MISSPELLINGS	Record misspellings and replacements in personal dictionary
-MISSPELLINGS	Do not record misspellings and replacements
QUIT	Quit checking this document
HELP or ?	Display this list of commands

Spelling Checker Commands

After receiving the command and any other necessary information, the checker can then proceed to execute the command and continue checking. When the checker reaches the end of the document, the user is asked if he wants to save his personal dictionary (if he has a personal dictionary).

Finally, the checker indicates it has finished by displaying "DONE" on the screen and returns. When checking an ETUDE document, the checker might move the document's cursor to the end of the region just checked or perhaps to its original position when it has finished checking; this could be in addition to displaying done or a substitute method of indicating that checking has been completed.

4.3. The Table System

This project involved creating a prototype table subsystem within the framework of an integrated office workstation. Tables are widely used in business, governmental, industrial and scientific communities for presenting information in a clear and concise manner. Because it is so widely used, a table system is a desirable component of an office workstation.

Our table system will have the integrated capabilities of interactive editing, formatting and mathematical derivation. The integration of all these capabilities is essential for general office use. Tables can be defined, formatted and inserted in documents all with one consistent set of commands.

Tables are presented in a variety of formats in different contexts, either out of necessity or to achieve greater clarity. For example, a user may want every entry in a table to have three significant digits after the decimal point because of the precision requirement of his data. Another user may want his data to be printed in the unit of thousands with trailing digits truncated although he may want to include the other digits in mathematical calculations. A user may want vertical lines to be inserted between major sections of the table for clearer organization. The user should be able to format a table interactively using a set of simple formatting commands.

Many entries in a business statistical table contain information derived from another part of the table. When one part of the table is changed, other parts of the table that are dependent on it need to be changed also. Automatic derivation of entries in the table from other parts of it is an essential feature of an automated system. Automatic derivation is achieved by storing the relationships between different parts of the table and triggering the re-evaluation of those relationships when the user changes an entry that defines other entries.

4.4. Page Layout

Brian Berkowitz worked on the formatting aspects of the ETUDE II system. He developed a functional specification for the formatting system and developed some prototype formatting software.

The functional specification proposed that ETUDE II be able to produce sophisticated documentation such as typeset technical reports and users manual. This requires that the system be able to:

- produce paginated documents consisting of multi-column pages. A user will be able to specify *constraints* associated with document structures (e.g., paragraphs, sections, quotations, and footnotes) indicating the situations in which it is inappropriate to break such document structures over column and page boundaries.

- automatically place figures and tables on pages according to page layout templates developed by the user or his/her organization. The figure placement algorithm will work closely with the page makeup algorithm to try to place a figure on the same page or column as its reference.

- handle complex table-like document structure consisting of several text and pictures components that are arranged spatially according to user desires (e.g., a picture, its caption, and credits or an entry in a sales catalogue).

- handle footnotes, indices, tables of contents, marginal notes, and references between document structures (e.g., a reference within a paragraph to a figure, footnote, or section).

As a first step toward implementing this formatter, we developed a line formatter; this module maintains the ETUDE II document as a continuous sequence of justified lines. The line formatter produces the data structure that can be used by the redisplay module to display the lines of text and also by the page makeup module to produce final document pages. The line formatter was designed to change the line structure in an incremental fashion, attempting to minimize the amount of computation necessary to re-format the text when an editing change is made. It can therefore be used to format text interactively.

We have also developed a prototype page makeup subsystem. This subsystem was designed to test out algorithms for breaking text into pages that include a mixture of text and figures. The prototype is batch oriented and transforms the data structure produced by the line formatter into a series of pages. The prototype is reasonably sophisticated and can produce either single or double-column paginated documents. It can handle footnotes as well as place figures referenced in the text using a library of predefined page layouts. The algorithm uses several different kinds of constraints in order to produce pages. It will not necessarily produce an optimal solution, but uses heuristic techniques to try to produce a compromise between the numerous and often conflicting constraints:

- page-break rules associated with the document structures (paragraphs, quotations, sections, etc.) that determine where it is appropriate to break such structures. An example rule is, "a paragraph should not be broken across a column unless it is at least five lines long and at least two lines of the paragraph appear at the bottom of the first column and at the top of the second column."

- balance rules indicating the desired depth of the page and the importance of setting pages at full depth and balancing columns,

- figure placement: These indicate the desired placement of a figure in terms of where on the page or *spread* (i.e., a pair of pages consisting of a left hand page and a right hand page when a document is printed double sided) the figure should be located

- desired spacing between document elements and the extent that the system is allowed to deviate from this desired spacing: Usually it is possible to slightly alter the vertical spacing between paragraphs (or even the spacing between lines) to produce balanced columns.

The algorithm is completely automatic at this stage. Our goal is to eventually produce a page formatter that can be used in an interactive fashion allowing the user to issue commands that manually alter the format of the page in order to reformat unattractive pages. We would like to enable the user to specify why the page was unattractive (e.g., because a figure was improperly placed or because a list with too few elements was broken). As the user specifies his desires to the page makeup algorithm by manually altering a few documents, the system will be able to better produce documents according to the desires of the user.

4.5. Command Parsing

ECOLE was designed as an operating system for the workstation; it provides essential services both to application programs and to the user. The command parser provides a common user interface that can be used by most applications, thus making the entire system easier to use: the user need learn only one set of conventions in order to take full advantage of any application's user interface. The sharing of code tends to produce better application programs: the application can be written without repeating the design of a good command parser.

The command parser that ECOLE provides was intended to work for many applications. As a result, it is almost entirely table-driven: with the exception of system-wide functions like **MENU**, no command is 'built-in.' In addition to parsing the user's input, the parser supervises the execution of commands; thus, most applications can be implemented without a top-level command loop. Rather, the application programmer needs to implement only the code for the individual application commands, ECOLE will ensure that the code for each command is called with suitable arguments.

The parser works off a hierarchy of three types of tables. The first, a character table, translates physical keystrokes on the workstation keyboard into logical characters recognized by the rest of the system. This approach frees the system from dependence on a particular terminal type or keyboard layout; if the system is being moved to a terminal with different keypads, one need only invent a new character table defining the mapping between the escape sequences sent by the terminal and the logical characters in the system--one does not have to write any new code to support the new keyboard layout. This approach has the additional advantage that several keys on the terminal can be made to map into the same logical character, and that each application can, if it chooses, change the definitions of keys on the terminal.

The second table type is by far the most complex. Command tables define the acceptable commands and command syntaxes for each application. The model used is pseudo-English imperative sentences: commands are assumed to be of the form

```
verb object object
```

. The command table contains verb descriptions, specifying the name of each verb and the type of each of its arguments. Thus, ETUDE might have a command

```
move region
cursor
```

. Each argument type is described by a syntax, which is a transition network where the condition for traversing an arc can be described by a function. Thus, application code can, if necessary, be called even during the parsing of a command.

The third table type is the dispatch table, which provides a mapping between operations and functions to execute them. This allows applications to share command tables while providing different services, or to provide the same service with different command tables.

ECOLE provides a framework for the implementation of applications by specifying when application code will be invoked and with what arguments. It is instructive here to consider a simple example. Let us suppose that ETUDE has been implemented using ECOLE, and that a user wants to say

```
goto end-of next chapter
```

. He will first type `goto`, which ECOLE will look up in the current command table; it will find a description of `goto` as a verb with one argument, a `location`. It happens that `end-of next chapter` is an acceptable description of a location, according to the location syntax; when a terminal node of the syntax is reached, ECOLE will invoke application code for the first time. It executes the `LOCATION` operation, with the characters typed by the user to specify a location: `end-of next chapter`.

The application will have a `LOCATION` operation defined in its dispatch table; the function specified there will examine the user's input. and the current state of the application, and return an object representing the location specified by the user. ECOLE will then notice that all the arguments for **GOTO** have been supplied; it will invoke the **GOTO** operation of the application, with the object returned by the **LOCATION** operation. This two-level approach to command parsing and execution imposes some structure on the application, and simplifies it in the process. That is, there must be separate code for specifying a location and for performing a goto; this makes it easier to implement other commands that also use locations.

Once the application has returned from its **GOTO** operation, ECOLE will invoke the **REDISPLAY** operation to update the application's part of the workstation screen; it will then invoke the **STATUS-LINE** operation to update the status display on the screen. Thus, the application can be assured that its display will always be current, without adding code in each command to update it.

This example has glossed over several issues, which we will discuss briefly here. First, one of the principles of ECOLE is that the user should not be constrained to use a particular form of input. If the user wishes, he can specify **GOTO** either by typing the special **GOTO** key, by spelling out 'goto', or by selecting Goto from a menu of all possible commands. This is true at any point in the parsing of a command: the user can always obtain a menu of all possibilities, and select from that; he can also spell out the names of special keys if he wishes (if, for example, he's unfamiliar with the layout of the particular keyboard he's using).

Second, at any point there is a hierarchy both of dispatch tables and of command tables. If the **STATUS-LINE** operation is not defined in the application's dispatch table, ECOLE will fall into the system default status line function. Similarly, system-level functions can be made available in all applications simply by including the system command table in the set of command tables, along with the application command table.

Finally, ECOLE attempts to be very flexible while providing a lot of support for the simple cases. Thus, arcs in syntaxes have programs associated with them in the most general case. They may, however, have simpler things: sets of logical characters, or sets of names, that must be typed in order to traverse the arc. In addition, general parsing routines have been written for the most common cases: there are predefined functions for parsing file names, numbers, and so on. As an example of the flexibility that is possible, an arc in a syntax can have a command table associated with it. Once ECOLE has decided that it wants to traverse that arc, it will start a recursive invocation of the command parser, using the command table associated with the arc as the primary command table. Commands from that table will be executed until one of them causes the recursive parser to terminate (another system service); at that point, parsing of the original command will resume.

Thus, the ECOLE command parser provides a powerful, flexible user interface for many applications. It is not yet in a final form: the **HELP** and **UNDO** functions have not been implemented, and there are some applications where it is not really suitable. The major flaw is that ECOLE requires all arguments to a command to be specified on the command line; a desirable alternative is to specify some of the arguments on the command line, and the rest by filling in a form.

4.6. Display Management

In the prototype version of ETUDE, the problem of organizing and displaying information on the screen was addressed in an ad-hoc manner. To satisfy our goal of a workstation that integrates different office applications and that allows the user to easily switch between applications, we needed a more systematic way of handling the physical display.

This is the job of the ECOLE display manager. The display manager provides output services for application programs, just as the ECOLE command parser provides input services. These services include:

- primitive output operations for displaying text (in multiple fonts) and graphics, and to mark points on the display with cursors

- a method for organizing information on the display

- mechanisms for efficiently updating the display image.

The central concepts in the display manager are the *virtual screen* and *window*. Virtual screens provide an interface between application programs and the physical display. A virtual screen serves three purposes. First, it defines a local coordinate system that is independent of its location on the physical screen. Second, output directed to a virtual screen is confined within its boundaries. Finally, a virtual screen hides from applications the low-level characteristics of the actual display (for example, the character sequences used to change fonts).

A window, on the other hand, represents the association between a virtual screen and an image of a data structure. Windows are used for organizing information on the display. The physical display is a limited resource that must be shared among the currently active subsystems. In addition, individual subsystems require a similar mechanism for organizing the information they want to present to the user.

In last year's report we described a prototype implementation of the display manager that was done on CLU. In that implementation, windows could be arranged hierarchically (i.e., one window could contain a series of other windows), and could overlap and obscure one another. The display manager automatically kept track of the part of each window that was currently visible on the display.

Various external factors caused us to switch languages from CLU to MDL, which required a new implementation of the display manager. The language switch was beneficial because it gave us the opportunity to reconsider the architecture of the display manager; feedback from users (notably, the PCAL implementors) greatly helped in this re-evaluation. The current (MDL) implementation shares some of the characteristics of the CLU version, although it is less ambitious in its capabilities and therefore more efficient and practical.

An important goal of the current display manager implementation is to avoid unnecessary and duplicate tests within the basic display manager code. For example, in the CLU implementation of the display manager, virtual screens automatically clipped output at their edges. In many applications, including ETUDE, information is always formatted before it is output; the goal of formatting is to fit the information into a given boundary, which is usually the same as the virtual screen. In these cases, any tests done during output will duplicate similar tests done during formatting. In the current implementation, therefore, clipping is enabled only when required by an application.

Another difference between the CLU and MDL versions of the display manager is that the latter has no inherent notion of overlapping windows. This means that applications are not unnecessarily burdened with the overhead associated with overlapping windows. Windows created by applications generally do not overlap; for example, ETUDE does not use overlapping windows to display the image of a document page. An application can, however, choose to overlap its windows, provided it manages the conflict between windows.

The process of updating a subsystem's window is handled using an ECOLE command. In the same way the the programmer specifies a function for performing a **MOVE** command, he also specifies a function for performing the **REDISPLAY** command. The command parser treats these commands identically, even though **REDISPLAY** cannot be directly invoked by the user. The programmer's redisplay function is given the window as an argument, from which it can access the data structure to be displayed and a virtual screen.

A redisplay function can be as simple or complicated as needed. Simple applications can just update the entire window. For applications (such as ETUDE) whose window changes only slightly after every command, an *incremental redisplay* function is appropriate. The purpose of incremental redisplay is to speed up the redisplay process and minimize the area of the display that changes after each command.

To help support incremental redisplay functions, each window contains a slot that programmers can use to store information about what is displayed in a window. This information, which can be in any format, is then used by the redisplay function to determine what parts of the window need to be updated.

4.7. Object Management

S. Zdonik has been working on the design of an object management system for integrated office workstation applications. An office workstation is a tool that is used by an office worker to produce objects that are required for the effective execution of their job. Examples of office object types are documents, graphics, calendars, and data tables. Modern workstations provide tools for producing various object types, but lack tools for the effective management of these objects. One reason for this void is that the characteristics of these *object management systems* have not as yet been clearly defined.

Traditional file systems provide facilities for the archival storage and retrieval of objects that are created in user programs. Database management systems provide a means of describing the semantics of a particular type of object, but the kinds of objects that they can handle are very inflexible. An object management system combines the advantages of both a file system and a database management system in that it can store arbitrarily defined programming language objects and at the same time maintain a high-level description of their meaning and intended behavior. The high-level description for all object types is in terms of a single model. The object management system can access this description, and, thereby, assist users in locating objects that meet their requirements.

The prob'em that is explored in this work is development of an appropriate model for defining the semantics of heterogeneous object types that are produced in an office environment. We begin by identifying the goals of an object management system:

1) **Object structure specification.** It is desirable to have a mechanism for describing the structure of the objects that are being managed by the object management system. A report might be described as containing a set of chapters, a set of appendices, and a bibliography. The object management system should also be able to access any of these object components, such as a single chapter of a report.

   The system would also accept descriptions of the objects that can appear as legal components of another object. For example, one could specify that a group project report can only contain chapters that have been written by members of the research group.

2) **Relationships among objects.** The object management system will store many objects of many different types. It is essential that the system have a means for expressing relationships among objects. A given paper might be related to some meeting object that represents a time when the authors will get together to discuss its content.

3) **Retrieval of objects.** An object management system, much like a database management system, provides a means for describing the properties of objects such that they can be used for retrieval later. In a database management system, users are often interested in performing retrievals such as *Get all employees who make more than $30K.* In an object management system, a user might ask for *all reports that are longer than 20 pages and that have been written by people who make more than $30K.*

4) **Effective memory utilization.** Another important function that an object management system should perform is the management of the way in which large objects are read into main memory. Since the system will have access to a large amount of information about the semantics of an object, we expect that it can do a good job of retrieving those pieces of a large object that are most likely to be needed.

5) **Object contrpl.** Object control is concerned with controlling the ways in which an object is used, including who is allowed to perform actions on objects and how multi-user action is to be managed. The user should have easy to use facilities for specifying these usage constraints.

6) **Side effects to change.** There must be a way to describe what actions are to be taken when some change happens.

7) **Alternative views of an object.** There should be a way to describe the construction of different objects based on the same underlying information base. An example of a different view of an object is the outward appearance of a document. Different outward appearances can be computed based on the ultimate output device.

The object management system must support the expression of high-level semantics of objects. The object semantics is described in terms of a high-level data model that we believe is flexible enough to express the meaning and behavior of a large class of office objects. The data model that we are proposing has facilities for describing four distinct aspects of objects. These aspects are:

Content	Objects can be made up of other objects. The components of an object are the other objects from which a given object is constructed. The content is the part of the object that is edited by one of the workstation subsystem programs. The content is also what gives the object its identity. Changing (i.e., editing) the content creates a new object.

Attributes	Attributes are like the fields of a record in a database management system. They are (object.attribute.value) tuples in which for a given *object* the *attribute* name has the value *value*. Changing an attribute does not create a new object; it simply updates a recorded property of the object.
History	The history of an object is the record of the different forms that a conceptual object takes on in the course of its evolution. For example, chapter one of May's monthly report may have been revised many times before it is accepted as a final product. The intermediate versions all represent the same conceptual entity, chapter one of May's report. The way in which this history is managed can be specified in the history aspect of an object-schema class definition.
Control	The control aspect relates to how an object can be used and what should happen when the object is used. The use of an object can include operations such as reading, writing, creating or updating one or more of its aspects. An example of the kind of control that one could specify would be the sending of a message to the author of a document whenever someone else creates a new version.

4.8. Extensions to MDL

Workstations should be able to operate in a heterogeneous environment. Two kinds of heterogeneity are relevant. *Logical heterogeneity* involves different specializations of the same basic idea. For example, sentences, sections, and text galleys can all be viewed as specializations of the concept of a sequence. *Physical heterogeneity* involves different computer representations of individual concepts. For example, lists, arrays, hash tables, and character strings are all possible implementations of sequences. Programs written to deal with heterogeneity can operate on a wider variety of objects, thus increasing the value of individual software components.

The ability of programs to deal with logical and physical heterogeneity is critically dependent on the host programming language. Since the MDL language was not designed with heterogeneity as a principal goal, D. Carnese extended the language to provide increased support for both types of heterogeneity. Five new capabilities were added: *generic procedures*, *classes and implementations*, *multiple coexisting implementations*, *implementable procedures*, and *subclasses*.

Generic procedures: A generic procedure is one which can execute different bodies of code depending on the type of one or more parameters. For example, a generic ADD procedure could invoke different procedures if its arguments were numbers, matrices, or columns of a table. Almost every programming language has built in generic operations (e.g., for +), but programmer-definable generics are much less common.

Definable generics are important for several reasons. Procedures can be more robust, since they can accept a wider variety of inputs. Program packages which define different kinds of objects can be combined more easily, since name conflicts among the packages can be resolved by making each procedure an alternative of a generic. And the name space of programs are reduced, since generic alternatives for different types are all invoked by the same name.

Classes and implementations:

The concept of "abstract types" has gained wide currency in the past decade. The essential idea is to use one kind of structure to represent the desired behavior of a group of objects, and another kind for the internal representation of the objects. In this way a "sequence" may be implemented in terms of "lists," "arrays" or other data structure.

The principal utility of classes and implementations is that programs can be designed to operate on any object which is an instance of a class, regardless of its implementation. Thus, changing the implementations of ECOLE objects does not require changing the application software which uses The result is that the objects that ECOLE programs manipulate can be physically heterogeneous over time.

A well-known problem with a simple class/implementation mechanism (e.g., such as the one pioneered by the Clu group) is that all instances of a class in a given environment are forced to use the same implementation. In order to further facilitate physical heterogeneity, we allow objects of the same class but different implementations to coexist. This facility is referred to as Coexisting Implementations.

Implementable procedures is a mechanism for allowing procedures to be written in terms of the most suitable of the co-existing implementations.

Subclasses: It is possible to define a class mechanism in which each class is logically independent from all others. However, it is often the case that the instances of one class can be considered as instances of other classes. For example, "manuals", "brochures", and "annual reports" can all be considered to be special cases of 'documents". As a consequence of the logical heterogeneity of the ECOLE environment, many such relationships exist between ECOLE classes.

Our class mechanism allows classes to be defined as *subclasses* of existing classes. If A is a subclass of B, all the operations of B are defined to be applicable to instances of A, and instances of A can be used wherever instances of B are appropriate. Unlike other languages which allow subclass-like structures (e.g., Smalltalk and Lisp Machine Lisp), the semantics of subclasses prescribe that for all operations of superclass B, the externally-observable behavior of instances of A and B should be identical.

4.9. Constraint-Based Graphics Editing System

It is natural for an advanced document preparation system to provide facilities for including non-textual information in a document. The most important non-textual information prevalent in traditional documents are graphs. With the advent of high-resolution bit-mapped displays as the preferred device for user-machine interaction and matrix printers for hardcopy generation, the need to interactively create graphs and compose them with text in a *unified* preparation environment is felt more than ever.

Bahram Niamir has been working on a language for representing graphical objects that would allow easy modification of such objects and would also support the integration of complex graphs with documents constructed by an ETUDE-like text editor in the process of document preparation.

We have identified three engineering principles in the construction of a graphics system for document preparation:

- **Structured Graphics** - The graphics system must allow for the creation of complex graphic objects whose parts are related both in topology and geometry to other parts of the object.

- **Interactive User Interface** - It is clear that the only natural and acceptable way to create and edit graphs is to manipulate the end object directly on the screen.

- **Text-Graphics Integration** - In order to fully integrate graphs with text or text within graphs (e.g., captions and legends), text and graphics must both be viewed as the same 'document,' albeit of different types.

Structured Graphics: A structured graphics system is one that contains inherent knowledge of the composition and interrelationships of graphic objects and their sub-parts. A structured graphics system maintains sufficient knowledge of the graphic object in order to present to the user the object in itself rather than some appearance of the object on the screen. The concept of a structured graphics editor is similar to that of a structured text preparation system where the document is structured and the editor uses the structure to edit, format, and provide an interface for the document to the user.

In a simple structured system, objects are represented not by their appearance (as done in an image editor) but by their geometry. That is, an object is a set of primitive points, lines, curves, and surfaces. Each of these parts may be individually selected, added, removed, or altered. Some systems structure their objects further by allowing for the clustering of primitive parts into sub-objects of their own and allowing for the naming of such sub-objects. Thus an object becomes a hierarchy of sub-objects. This hierarchy may be used in the process of creating objects and the selection and manipulation of an object's parts. Further structuring may be attained by sharing of sub-parts between objects (resulting in non-tree like objects) and by having object *types* from which individual objects are instantiated.

A graphic object also has a topology. The topology of an object is the relationship among the points of the object, irrespective of the size, location, or inclination of the object (the object's geometry). For example, a parallelogram is topologically determined by the property that opposite pairs of vertices are equidistant. A structured system must be able to represent object topology irrespective of the object geometry.

The language we use to represent an object's topology and geometry is based on a system of simultaneous equations. Each equation (also known as a *constraint*) signifies a relationship among the points and vectors of the object. In a parallelogram, the variables in the constraint equation are the four vertices and two adjacent sides. An appropriate system of 3 constraints among these variables determines a parallelogram.

In addition to its system of constraints, an object also has an *appearance*. The appearance of an object are the actual lines, curves, and surfaces drawn on the screen as bounded and determined by the variables of the object. In order for an object to have an appearance, its system of constraints must be fully determined. The variables of an object so not necessarily appear in its drawing. However, the topology of an object need not be visible and discernible by its appearance.

A system of constraints is procedural in the sense that the value of a variable is algorithmically determined from the values of other variables, and the structure of an object is determined by calling sub-objects and passing constraints as parameters to them. A system of constraints is declarative in the sense that it represents topology and geometry without specifying how the object is to be constructed. A distinct advantage of constraints is the non-procedurality. The identity of an object does not depend upon the order by which the constraints are defined to reach at the object, neither does it matter in what order the sub-objects of the object are defined to reach at the object. The non-procedurality of systems of constraints make them particularly attractive to interactive manipulation.

Interactive User Interface: An interactive graphics system allows that the graphic object to be directly manipulated in terms of its end representation on the screen rather than to its source specification. In our constraint-based system, the user will create, select, and alter an object by issuing interactive commands that appear to manipulate the end representation on the screen. However, these commands will be translated into internal operations that insert, delete, or modify the constraints of the object. The user may also issue commands that alter the *appearance* of the object without affecting its constraints.

Altering the constraints of an object is conceptually extremely simple. The old constraint is deleted, the new constraint is added, and the system is re-determined for the new object. If the new constraint over-determines the system, a warning is issued and the command is ignored. It is expected that many structure editing commands may be categorically translated into simple constraint insertion and deletion operations. Also, by the fact that the system of constraints is non-procedural, a change to an object will propagate only to other objects that depend on the first object. In a procedural language, the user is often forced to create objects in a specific order which makes later objects depend upon earlier objects (in size, location, etc.). A change to an object will affect all succeeding objects even if the user did not intend such a dependency.

Objects that are not fully determined can also be drawn if the user creates *default* constraints that will fully determine the object. Such *default* constraints exist only to provide the object with an appearance. *Default* constraints are always overridden if other constraints are entered that make the system over-determined. Hence an under-determined object may still be drawn and interactively altered even though it represents a family of objects. For example, if the constraint governing the height of a rectangle is made into a *default* constraint, the rectangle will have an undetermined height, but will still have an appearance for further interactive manipulation.

Text-Graphics Integration: The major issues in text-graphics integration are as follows:

- Two-dimensional page layout and the spatial integration of text inside a document and a graph created by a constraint-based editor. Our approach here is also constraint based. Each graph will be constrained within a rectangular *frame* and the page layout system must know about the concept of a *frame* and be prepared to modify the layout as the frame changes.

- Text within graphs (captions and legends) must be set by the full power of the text editor. Hence text must be treated as text rather than as a special kind of graphical object. Again, the vehicle of integration will be rectangular frames to contain the text.

- The command interface to the graphics editor must conceptually mimic the command interface to the text preparation system.

- Cross-referencing between text in separate frames, or between graphic objects in separate frames must be allowed. For example, the height of a bar in a bar graph might depend on the text entered in a table entry elsewhere.

Publications

1. Good, M.D. "An Ease of Use Evaluation of an Integrated Document Processing System," In Proceedings of Human Factors in Computer Systems, Gaithersburg, Maryland, March 15-17, 1982, pp. 142-147. Also available as Office Automation Group Memo OAM-036, January 1982.

2. Good, M.D. "An Ease of Use Evaluation of an Integrated Editor and Formatter," MIT/LCS/TR-266, MIT Laboratory for Computer Science, Cambridge, MA, November 1981. Revised version of S.M. Thesis.

3. Good, M.D. "How to Use ETUDE," MIT Office Automation Group Memo OAM-034, Cambridge, MA, with Eric Munro.

4. Greif, I. "Computer Support for Cooperative Office Activities," In Proceedings of the 1982 Office Automation Conferen ce, San Francisco, California, April 1982, AFIPS Press.

5. Greif, I. "Cooperative Office Work, Teleconferencing and Calendar Management: A Collection of Papers," MIT/LCS/TM-218, MIT Laboratory for Computer Science, Cambridge, MA, December 1981.

6. Greif, I. "PCAL: A Personal Calendar," MIT/LCS/TM-213, MIT Laboratory for Computer Science, Cambridge, MA, December 1981. Office Automation Group Memo.

7. Greif, I. "Teleconferencing and the Computer-Based Office Workstation," In Teleconferencing and Interactive Media '82, Madison, Wisconsin, May 19-21, 1982.

8. Greif, I. "The User Interface of a Personal Calendar Pıogram," In Proceedings of the NYU Symposium on User Interfaces, New York University, New York, N.Y., May 26-28, 1982.

9. Sirbu, M., Shoichet, S., Kunin, J., Hammer, M. and Sutherland, J. "OAM: An Office Analysis, Methodology," In Proceedings osf the 1982 Office Automation Conference, April 1982, AFIPS Press.

Theses completed

1. Bauman, B.D. "A Distributed Database Facility for a Personal Calendar System," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1982.

2. Blake, L. "Attitudinal Effects of Office Automation on the Workers at I.L.P.," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1982.

3. Czarnecki, P.A. "Semcal: Avoiding Data Deluge in the Office Environment," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1982.

4. Good, M.D. "An Ease of Use Evaluation of an Integratedd Editor and Formatter," S.M. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, August 1981.

5. Hsu, K.H. "*Sharing of an Office Calenddar*," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1982.

6. Kim, Y. "Resource Sharing in an Automated Calendar System - PCAL," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1982.

7. Mok, W.N.K. "The Design, Implementation and Integration of a Table System," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

8. Mondori, A. "Evaluation and Modification of Calendar's Terminal Interface," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

9. Nitchman, J.E. "An Interactive Spelling Checker and Corrector for ETUDE," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1982.

10. Rosenstein, L.S. "Display Management in an Integrated Office Workstation," S.M. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, February 1982.

11. Tallian, A.E. "Match: A Query System for the Personal Calendar," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1982.

12. Zurko, M.E. "Protection Numbers on XX: from Obscurity to Clarity," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1982.

Theses in Progress

1. Adamoli, A.J. "Simultaneous Joint Editing of Text Documents," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, Expected August 1982.

2. Carnese, D.J. "The Simplicity/Performance Conflict in Type Definition," S.M. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, Expected August 1982.

3. Niamir, B. "Constraint-Based Interactive Graphics Editing for Document Preparation," Ph.D. Dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, Expected December 1982.

4. Sarin, Sunil D. "Remote Meeting Support Through Interactive Sharing of Computer-based Information," Ph.D. Dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, Expected December 1982.

5. Zdonik, S.B. "An Object Management System for Integrated Office Workstation," Ph.D. Dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, Expected January 1983.

Talks

1. Good, M.D. "An Ease of Use Evaluation of an Integrated Document Processsing System," Digital Equipment Corporation Human Factors Steering Group, Hudson, MA, March 3, 1982.

2. Good, M.D. "Human Factors in Computer Systems," Gaithersburg, Maryland, March 15, 1982.

3. Greif, I. "Computer Support for Group Activities," MIT Laboratory for Computer Science, Cambridge, MA, November 24, 1981.

4. Greif, I. "Cooperative Office Activities: Calendar Management and Desk-to-Desk Conferencing," Exxon Office Systems, Princeton, NJ, June 8, 1982.

5. Greif, I. "Cooperative Office Activities," ILP Symposium on Developing and Implementing a Corporate Office Automation Program, May 26, 1982.

6. Greif, I. "Cooperative Office Activities," Office Automation Conference, San Francisco, CA, April 5, 1982.

7. Greif, I. "OAM and OSL: An Office Automation Methodology (a two day course)," The Israel Institute of Productivity, Israel Centre for Information Systems Center for Productivity, Tel Aviv, Israel, July 1981 and "Research in Office Automation at the MIT Laboratory for Computer Science."

8. Greif, I. "Research in Office Automation at MIT Laboratory for Computer Science," Apple Computer, Cupertino, CA, August 1981.

9. Greif, I. "Update on Research in Office Automation at LCS," Office Automation Roundtablel, DARCOM, April 22, 1982.

10. Greif, I. "The User Interface of a Personal Calendar Program," NYU Symposium on User Interfaces, New York University, New York, NY, May 26-28, 1982.

11. Sirbu, M. "Advances in Electronic Mail," Pergammon Infotech State of the Art Tutorial, London, England, October 9-11, 1981.

12. Sirbu, M. "Integrating Personal Computers, Word Processing and Timesharing," 1982 Office Automation Conference, April 1982.

13. Sirbu, M. "Office Automation and Communications," Technology Education Associates Seminar, Sydney, Australia, July 13-15, 1981.

14. Sirbu, M. "Understanding Managerial Group Work," 1982 Office Automation Conference, April 1982.

15. Ilson, R. "Document Processing Languages," Integrating CAD/CAM and Electronic Publishing - Sponsored by the Graphic Communications Association, April 1982.

16. Ilson, R. "Systems for Enhancing Human Information Exchange Series - Etude System - (two talks)," IBM Systems Research Institute, December 1981.

17. Zdonik, S.B. "Logical Design of Databases," Keynote Address to DRS Users Group, Boston, MA, November 1981.

18. Zdonik, S.B. "Panel on Office Database Systems," Panel member, SIGMOD '81, May 1981.

PROGRAMMING METHODOLOGY

Academic Staff

B. H. Liskov, Group Leader

Visitors

N. A. Lynch	J. E. Stoy

Research Staff

P. R. Johnson	R. W. Scheifler

Graduate Students

T. Bloom	B. M. Oki
S. Y. Chiu	J. C. Schaffert
J. A. Goree, Jr.	E. W. Stark
C. Henderson	E. F. Walker
M. P. Herlihy	W. E. Weihl
J. N. Lancaster	

Undergraduate Students

W. Berger	B. O'Connor
M. DuLong	

Support Staff

A. Rubin

1. INTRODUCTION

This year we have continued our work on linguistic support for the construction and execution of distributed programs. We have focused on the development of a new language and its support system. This new language is now called Argus; it is a substantial extension of CLU [1], with new features to support concurrency, distribution, and reliability. Argus is described in [2].

During the current year we have concentrated on completing the design of Argus. As part of this effort, we have looked at example distributed applications to evaluate the expressiveness of various features. We show one such example below. Our tentative conclusion based on such examples is that Argus provides a good framework for distributed programs. Of course, we must wait for Argus to be implemented, and used in constructing real applications, before we can make a more informed judgment.

In addition to our work on the Argus design, we have also prepared for the Argus implementation. Our initial implementation will be on Vax's running Unix and communicating over an ethernet. A major step in the Argus implementation is moving CLU to the Vax. This move has been accomplished, and an implementation of CLU on the Vax under Unix is now available for distribution.

In conjunction with our work on the Argus design and implementation, we have also been studying the semantic foundations of Argus. For example, Argus makes use of nested atomic actions. Precise definitions of the semantic concepts in Argus are needed, both to support reasoning about Argus programs, and to support reasoning about the correctness of the Argus implementation.

The remainder of this section is organized as follows. In Section 2 we give a summary of the major features of Argus. Section 3 contains an example Argus program and a discussion of the expressive power of Argus. Finally, in Section 4 we describe a portion of the semantic model of nested actions.

2. SYSTEM OVERVIEW

In Argus, a distributed program is composed of a group of *guardians* running on *nodes* (computers) connected (only) via a communications network. A guardian encapsulates and controls access to one or more resources. The external interface of a guardian consists of a set of operations called *handlers*, which may be invoked by other guardians. The guardian executes the calls on these handlers, synchronizing them as needed. Furthermore, it may refuse to perform an access desired by a caller if the caller does not have proper authorization. Handler invocation is performed using a message-based communication mechanism with *at-most-once* semantics: either the request is delivered and acted on exactly once, with exactly one reply received, or the request is never delivered and the sender is so informed.

Internally, a guardian contains data objects and processes. The processes perform handler calls (there is a separate process for each call) and background tasks. Some of the data objects comprise the global state of the guardian; these objects, such as the actual resources, are shared by the processes. Other objects are local to the individual processes.

A guardian's global state may consist of both *stable* and *volatile* objects. Stable objects are written periodically to *stable storage devices*. These are devices that, with very high probability, do not lose the information entrusted to them [3]. After a crash of the guardian's node, the language support system re-creates the guardian with the stable objects as they were when last written to stable storage, as discussed further below. A process is started in the guardian to re-create the volatile objects. Once the volatile objects have been restored, the guardian can resume background tasks, and can respond to new requests.

Although the processes inside a guardian can share objects directly, direct sharing objects between processes in different guardians is not permitted. The only method of inter-guardian communication is by invoking handlers, and the arguments to handlers are passed by value: it is impossible to pass a reference to an object in a message.

In almost any system where on-line data is being read and modified by on-going activities, there are important consistency constraints that must be maintained. Such constraints apply not only to individual pieces of data, but to distributed sets of data as well. The main issues here are the coordination of concurrent activities (permitting concurrency but avoiding interference), and the masking of hardware failures.

The distributed state of a system is a collection of data objects that reside at various guardians in the network. Some of these objects are stable, for which the probability of loss of information due to hardware failures is extremely small. Other objects are volatile, with a relatively high probability of loss, and as such must contain only redundant information if the system as a whole is to avoid loss of information.

An activity can be thought of as a process that attempts to examine and transform some objects in the distributed state from their current (initial) state to some new (final) state, with any number of intermediate state changes. Our approach to maintaining consistency is to make activities *atomic*. Two properties distinguish an activity as being atomic: indivisibility and recoverability. By indivisibility, we mean that the execution of one activity never appears to overlap (or contain) the execution of any other activity. By recoverability, we mean that the overall effect of the activity is all-or-nothing: either all of the objects remain in their initial state, or all change to their final state. If a failure occurs while an activity is running, either it must be possible to complete the activity, or to restore all objects to their initial states.

We call an atomic activity an *action*. An action may complete either by *committing* or *aborting*. When an action aborts, the effect is as if the action had never begun: all modified objects are restored to their previous state. When an action commits, all modified objects take on their new states; only at this point do changes to stable objects become permanent.

Actions can be *nested*. An action may contain any number of subactions, some of which may be performed sequentially, some concurrently. This structure cannot be observed from outside: the overall action still satisfies the atomicity properties. Subactions appear as atomic activities with respect to other subactions of the same parent. Subactions can commit and abort independently, and a subaction can abort without forcing its parent action to abort. However, the commit of a subaction is conditional: even if all subactions commit, aborting the parent action will abort all of the subactions. Further, changes to stable objects become permanent only when top-level actions commit.

To implement atomicity, we need to synchronize access to shared objects, and we need to be able to undo the changes made to objects by aborted actions. Such synchronization and recovery are not provided for all objects. For example, objects that are purely local to a single action do not require these properties. The objects that do provide these properties are called *atomic objects*, and atomicity is guaranteed only when the objects shared by actions are atomic objects.

Atomic objects are encapsulated within *atomic* abstract data types. Atomic types have operations just like normal data types, except that operation calls provide indivisibility and recoverability for the calling actions. Argus provides, as built-in types, atomic arrays, records, and variants, with operations nearly identical to the normal arrays, records, and variants provided in CLU. In addition, objects of built-in scalar types, such as characters and integers, are atomic, as are structured objects of built-in *immutable* types, such as strings, whose components cannot change over time. Users can also define new atomic types.

Our implementation of built-in atomic objects is based on a read/write locking model. Before an action uses an object, it must acquire a lock in the appropriate mode. To keep the locking rules simple, we do not allow a parent action to run concurrently with its children. An action may obtain a read lock on an object provided every action holding a write lock on that object is an ancestor. An action may obtain a write lock on an object provided every action holding a (read or write) lock on that object is an ancestor. When a write lock is obtained, a volatile *version* of the object is made, and the action operates on this version.

When an action aborts, its locks and versions are discarded. When a subaction commits, its locks and versions are inherited by its parent. When a top-level action commits, its locks are discarded but its versions become the new values of their respective objects, and any new versions of stable objects are written to stable storage, to be used as the initial versions following a node crash. To ensure global consistency, a two-phase commit protocol is used for top-level actions. In the first phase, an attempt is made to verify that all locks are still held, and to record the new version of each modified stable object on stable storage. If the first phase is successful, then in the second phase the locks are released, the recorded versions become the current versions, and the previous versions are forgotten. If the first phase fails, the recorded versions are forgotten and the action is forced to abort, restoring the objects to their previous states.

3. A SIMPLE MAIL SYSTEM

In this section we present a very simple mail system. Although we have chosen inefficient implementations for some features, and have omitted many necessary and desirable features of a real mail system, we hope to give some idea of how a real system could be implemented in Argus.

The interface to the mail system is quite simple. Every user has a unique name (*user_id*) and a mailbox. However, mailbox locations are completely hidden from the user. Mail can be sent to a user by presenting the mail system with the user's *user_id* and a *message*; the message will be appended to the user's mailbox. Mail can be read by presenting the mail system with a user's *user_id*; all messages are removed from the user's mailbox and are returned to the caller. For simplicity, there is no protection on this operation: any user may read another user's mail. Finally, there is an operation for adding new users to the system, and some operations for dynamically extending the mail system.

All operations are performed within the action system. For example, a message is not really added to a mailbox unless the sending action commits, messages are not really deleted unless the reading action commits, and a user is not really added unless the requesting action commits.

The mail system is implemented out of three kinds of guardians: *mailers*, *maildrops* and *registries*. Mailers act as the front end of the mail system: all use of the system occurs through calls of mailer handlers. To achieve high availability, many mailers will exist, e.g., one at each physical node. A maildrop contains the mailboxes for some subset of users. Individual mailboxes are not replicated, but multiple, distributed maildrops are used to reduce contention and to increase availability, in that the crash of one physical node will not make all mailboxes unavailable. The mapping from *user_id* to maildrop is provided by the registries. Replicated registries are used to increase availability, in that at most one registry need be accessible to send or read mail. Each registry contains the complete mapping for all users. In addition registries keep track of all other registries.

Figure 1 defines a number of abbreviations for atomic types used in implementing the mail system. For simplicity, we use only types obtained from the built-in atomic type generators *struct* and *atomic_array*, together with the abstract types *user_id* and *message*, whose implementations we omit. Structs are *immutable* records: new components cannot be stored in a struct object once it is built. Since structs are immutable, they are atomic. Atomic arrays are one-dimensional, and can grow and shrink dynamically. Of the array operations used in the mail system, *new* creates an empty array, *addh* adds an element to the high end, *trim* removes elements, *elements* iterates over the elements from low to high, and *copy* copies an array. Read locks on the entire array are obtained by *new*, *elements*, and *copy*, and write locks are obtained by *addh* and *trim*.

The mailer guardian definition is presented in Figure 2. The **is** clause lists the various ways of creating instances of the guardian; all of the guardians presented in this section have a single *create* operation. The **handles** clause lists the externally available handlers of the guardian. Then follows the stable and volatile variables comprising the global state of the guardian. The **recover** code is used to reinitialize the volative variables after a crash. The **background** code is used to perform any background tasks needed by the guardian. Finally there are the implementations of the creation operations and the handlers.

Figure 9-1: Abbreviations

```
mailbox      = struct[ mail: messagelist,     % messages for
                       user: user_id]         % this user
messagelist  = atomic_array[message]
mailboxlist  = atomic_array[mailbox]
registrylist = atomic_array[registry]
steeringlist = atomic_array[steering]
steering     = struct[ users: userlist,       % users with mailboxes
                       drop: maildrop]        % at this maildrop
userlist     = atomic_array[user_id]
```

Figure 9-2: Mailer Guardian

```
mailer = guardian  is create
                   handles  send_mail, read_mail,
                            add_user, add_maildrop, add_registry

stable some: registry       % stable handle
best: registry              % volatile handle

recover
   best := some             % reassign after crash
   end

background
   while true do
      enter topaction
         best := ... % choose closest responding registry
         end
      sleep(...)
      end
   end

create = creator (reg: registry) returns (mailer)
   some := reg
   best := reg
   return(self)
   end create

send_mail = handler (user: user_id, msg: message)
                            signals (no_such_user)
   best.lookup(user).send_mail(user, msg)
      resignal no_such_user
   end send_mail

read_mail = handler (user: user_id)  returns (messagelist)
                                     signals (no_such_user)
   return(best.lookup(user).read_mail(user))
      resignal no_such_user
   end read_mail
```

```
add_user = handler (user: user_id) signals (user_exists)
    drop: maildrop := best.choose()
    all: registrylist := best.all_registries()
    coenter
      action
          drop.add_user(user)
      action foreach reg: registry in registrylist$elements(all)
          reg.add_user(user, drop)
             abort resignal user_exists
      end
    end add_user

add_maildrop = handler ()
    all: registrylist := best.all_registries()
    drop: maildrop := maildrop$create()
    coenter action foreach reg: registry in registrylist$elements(all)
        reg.add_maildrop(drop)
        end
    end add_maildrop

add_registry = handler ()
    all: registrylist := best.all_registries()
    new: registry := registry$create(all, best.all_steerings())
    coenter action foreach reg: registry in registrylist$elements(all)
        reg.add_registry(new)
        end
    end add_registry

end mailer
```

A mailer must be given a registry when created; this registry is the mailer's stable "handle" on the entire mail system. The mailer also keeps a volatile handle: the registry representing the "best" access path into the system. The **background** code is used to periodically choose a new registry to play this role; the closest responding registry would be an appropriate choice.

A mailer performs a request to send or read mail by first using the *best* registry to determine the maildrop of the specified user, and then forwarding the request to that maildrop. The syntax for invoking a handler of a guardian is

```
expression.handler_name(arguments)
```

where the expression evaluates to a guardian object. A mailer adds a new user by first using the *best* registry to choose a maildrop, and then concurrently asking maildrop to create a mailbox and informing all registries of the new user/maildrop pair. The **coenter** statement is used to create concurrent subactions; the **foreach** clause indicates that a separate subaction should be spawned for each registry in the list *all*. Note that if the user is discovered to exist at any registry, the overall action aborts.

A new registry is added by extracting the entire user-to-maildrop mapping and the list of all registries from the *best* registry, and using them to create a new registry. The other registries are then informed of the new registry so they may add it to their registry lists. Finally, a new maildrop is added by creating one and informing all registries of its existence.

Figure 3 shows an implementation of the registry guardian. The state of a registry consists of an array of registries, together with a *steering list* associating an array of users with each maildrop. When a registry is created, it is given an array of all other registries, to which it adds itself, and the current steering list. The *add_user* handler checks to make sure the user is not already present, and adds the user to the user array for the given maildrop. The *add_maildrop* and *add_registry* handlers perform no error-checking because correctness is guaranteed by the mailer guardian.

An implementation of the maildrop guardian is given in Figure 4. The state of a maildrop consists of an array of mailboxes; a mailbox is represented by a struct containing a *user_id* and an array of messages. A maildrop is created with no mailboxes. The *add_user* handler can be invoked to add a mailbox. Note that this handler does not check to see if the user already exists; this checking is performed by the registries. The *send_mail* and *read_mail* handlers use linear search to find the correct mailbox. When the mailbox is found, *send_mail* appends a message to the end of the message array; *read_mail* first copies the array, then deletes all messages, and finally returns the copy. Both handlers assume the user exists; this is guaranteed by the registries.

Finally, in Figure 5, we show a simple use of the mail system, namely, sending a message to a list of users, with the desire that the message be delivered only if all of the users exist, and otherwise to get back a list of all non-existent users. The message is sent to all of the users simultaneously, and the non-existent users are collected in an array. Although a non-atomic array is used, its *addh* operation is defined to be indivisible, so no explicit synchronization is needed here. After all sends are completed, if the array is non-empty, the overall action is aborted, thus ensuring that none of the users are sent mail.

Figure 9-3: Registry Guardian

```
registry = guardian  is create
                     handles  lookup, choose, all_registries, all_steerings,
                              add_user, add_maildrop, add_registry

stable registries: registrylist      % all registries
stable steerings: steeringlist       % all users and maildrops

create = creator (rest: registrylist, steers: steeringlist) returns (registry)
   registrylist$addh(rest, self)  % add self to list
   registries := rest
   steerings := steers
   return(self)
   end create

lookup = handler (user: user_id)  returns (maildrop)
                                  signals (no_such_user)
   for steer: steering in steeringlist$elements(steerings) do
      for usr: user_id in userlist$elements(steer.users) do
         if usr = user
            then return(steer.drop) end
         end
      end
   signal no_such_user
   end lookup

choose = handler () returns (maildrop) signals (none)
   if steeringlist$empty(steerings)
      then signal none end
   drop: maildrop := ... % choose, e.g., maildrop with least users
   return(drop)
   end choose

all_registries = handler () returns (registrylist)
   return(registries)
   end all_registries

all_steerings = handler () returns (steeringlist)
   return(steerings)
   end all_steerings
```

```
add_user = handler (user: user_id, drop: maildrop) signals (user_exists)
    for steer: steering in steeringlist$elements(steerings) do
        for usr: user_id in userlist$elements(steer.users) do
            if usr = user
                then signal user_exists end
            end
        if steer.drop = drop
            then userlist$addh(steer.users, user) end   % append user
        end
    end add_user

add_maildrop = handler (drop: maildrop)
    steeringlist$addh(steerings, steering${ users: userlist$new(),
                                            drop: drop})
    end add_maildrop

add_registry = handler (reg: registry)
    registrylist$addh(registries, reg)
    end add_registry

end registry
```

3.1. Remarks

Close examination of the mail system will reveal many places where the particular choice of data representation leads to far less concurrency than might be expected. For example, in the maildrop guardian, since both *send_mail* and *read_mail* modify the message array in a mailbox, either operation will lock out all other operations on the same mailbox until the executing action commits to the top level. Even worse, since both *send_mail* and *read_mail* read the mailbox array, and *add_user* modifies that array, an *add_user* operation will lock out all operations on all mailboxes at that maildrop. In the registry guardian, an *add__user* operation will lock out *lookup* operations on all users with mailboxes at the given maildrop, and an *add_maildrop* operation locks out all *lookup* operations.

In a real system, this lack of concurrency would probably be unacceptable. What is needed are data types that allow more concurrency than simple atomic arrays. For example, an associative memory that allowed concurrent insertions and lookups could replace the mailbox array in maildrops and the steering list in registries; a queue with a "first-commit first-out" semantics, rather than a "first-in first-out" semantics, could replace the message arrays in maildrops. Such types can be built as user-defined atomic types, although we will not present implementations here.

Figure 9-4: Maildrop Guardian

```
maildrop = guardian is create handles send_mail, read_mail, add_user

stable boxes: mailboxlist := mailboxlist$new()

create = creator () returns (maildrop)
    return(self)
    end create

send_mail = handler (user: user_id, msg: message)
    for box: mailbox in mailboxlist$elements(boxes) do
        if box.user = user
            then messagelist$addh(box.mail, msg)  % append message
                return
            end
        end
    end send_mail

read_mail = handler (user: user_id) returns (messagelist)
    for box: mailbox in mailboxlist$elements(boxes) do
        if box.user = user
            then mail: messagelist := messagelist$copy(box.mail)
                messagelist$trim(box.mail, 1, 0)  % delete messages
                return(mail)
            end
        end
    end read_mail

add_user = handler (user: user_id)
    mailboxlist$addh(boxes, mailbox${ mail: messagelist$new(),
                                      user: user})
    end add_user

end maildrop
```

The concurrency that *is* built in to the mail system leads to a number of potential deadlock situations. For example, in the registry guardian, two instances of *add_user* could simultaneously read the same user array, and then simultaneously attempt to modify that array, neither succeeding because the other still holds a read lock. In the mailer guardian, deadlock is possible if two different *add__user*, *add_maildrop*, or *add_registry* requests modify registries in opposite orders.

Figure 9-5: Distributing Mail

```
distribute_mail = proc (m: mailer, users: idlist, msg: message)
                                signals (no_such_users(idlist))
   idlist = array[user_id]
   enter action
      bad: idlist := idlist$new()
      coenter action foreach user: user_id in idlist$elements(users)
         m.send_mail(user, msg)
           except when no_such_user:
                                idlist$addh(bad, user)      % indivisible
            end
         end
      if ~idlist$empty(bad)
         then abort signal no_such_users(bad) end
      end
   end distribute_mail
```

Some of these deadlock situations would go away if data representations allowing more concurrency were used. For example, the use of a highly concurrent associative memory for the steering list would allow *add_maildrop* requests to run concurrently. In other cases, the algorithms must be modified. For example, to avoid a deadlock between two different requests to add the same user, the mailer *add_user* operation could pick a distinguished registry, such as the first one in the list of all registries, and perform the registry *add_user* operation there sequentially before performing all of the rest concurrently. To avoid deadlock between concurrent *add__maildrop* and *add__registry* requests, the mailer *add__registry* operation could first get a write lock on the registry list of a distinguished registry, and *add_maildrop* could be forced to obtain its registry list from that same registry.

It may be argued that the strict serialization of actions enforced by the particular implementation we have shown is not important in a real mail system. This does not mean that actions are inappropriate in a mail system, just that the particular granularity of actions we have chosen may not be the best. For example, if an action discovers that a user does (or does not) exist, it may not be important that the user continues to exist (or not exist) for the remainder of the overall action. It is possible to build such "loopholes" through appropriately defined abstract types. As another example, it might not be important for all registries to have the most up-to-date information, provided they receive all updates eventually. In particular, when adding a new user, it may suffice to guarantee that all registries eventually will be informed of that user. This could be accomplished by keeping appropriate information in the stable state of a mailer guardian, and having the background process of that mailer be responsible for eventually informing all registries.

4. THEORETICAL STUDIES: CONCURRENCY CONTROL FOR RESILIENT NESTED ACTIONS

4.1. Motivation

Argus is based in part on a nested transaction model of programming which generalizes the single-level transaction model often used for databases. (See [4], for example.) Thus, the implementation of Argus poses problems similar to (and more difficult than) those which arise in implementing single-level transaction systems. An entire field of research, "concurrency control theory," is directed towards understanding the requirements and implementation of single-level transaction systems. We are generalizing this theory to handle nested transactions.

The concurrency control problem is roughly as follows. Data in a large (centralized or distributed) database is assumed to be accessible to users via *transactions*, each of which is a sequential program which can carry out many steps accessing individual data objects. It is important that the transactions appear to execute "atomically," i.e., without intervening steps of other transactions. However, it is also desirable to permit as much concurrent operation of different transactions as possible, for efficiency. Thus, it is not generally feasible to insist that transactions run completely serially. A notion of *equivalence* for executions is defined, where two executions are equivalent provided they "look the same" to all transactions and to all data objects. The *serializable* executions are just those which are equivalent to serial executions. One goal of concurrency control design is to insure that all executions of transactions be serializable.

Several characterization theorems have been proven for serializability; generally, they amount to the absence of cycles in some relation describing the dependencies among the steps of the transactions. A very large number of concurrency control algorithms have been devised. Typical algorithms are those based on two-phase locking [4] and those based on timestamps [5], [6]. Although many of these algorithms are very different from each other, they can all be shown to be correct concurrency control algorithms. The correctness proofs depend on the absence-of-cycles characterizations for serializability.

The nested-transaction model generalizes the earlier model in two important ways: by the nesting structure and by explicit consideration of aborts.

At first glance, the basic correctness criteria for nested transactions seem to be clear enough (intuitively) to allow implementors a sufficient understanding of the requirements for their implementation. However, some subtle issues of correctness arise in connection with the behavior of failed sub-transactions. For example, a pleasant property for an implementation to have is that "orphans" (sub-transactions of failed transactions) should see "consistent" views of the data (i.e., views that *could occur during an execution* in which they are not orphans). We go to considerable lengths to insure this property, but it is difficult for us to be sure that we have succeeded.

In order for us to be able to prove that our implementation satisfies its correctness requirements, it seems that we must generalize concurrency control theory to incorporate consideration of nesting and aborts.

4.2. Work Completed

We have begun to generalize the basic theory of concurrency control to treat nested transaction with aborts. In this section, we outline the steps which have already been accomplished.

Action Trees: First, we have defined a simple "action tree" structure. This structure describes the ancestor relationships among executing transactions and also describes the views which different transactions have of the data.

Formally, we assume a priori existence of a universal tree of possible transactions, hereafter called "actions", with fixed designation of ancestor relationships and sequential dependencies among actions. A (virtual) action U, the parent of all top-level actions, has been added for the sake of uniformity. The leaves of this tree of actions are called *Accesses*, and model program steps which actually access data objects. The objects they access and the functions performed on those objects are assumed to be fixed a priori. In any particular execution, only some of these possible actions will be "activated."

An *action tree* describes a snapshot of an execution. It consists of a subset of the set of all actions, together with *status* information ('active,' 'committed' or 'aborted') for each non-access action and a *label*, indicating data value read, for each access.

We have defined the important concept of *visibility* of one action in an action tree, to another. One action, A, is visible to another action, B, if A is committed up to the least common ancestor of A and B. That is, A is not hidden from B by an intervening failure or still-active action.

We have defined the notion of "serializability" for action trees. It amounts to the existence of a total ordering for each set of siblings in the tree, such that all the data values seen by the accesses are consistent with execution of the siblings in the given order. This consistency specifies that an access, B, only sees the effects of preceding accesses to the same object which are visible to B.

The most basic correctness requirements for nested transaction systems involve only those actions whose effects are "permanent" (formally, those which are visible to U). Thus, for any action tree T, we have defined the permanent part of T, *perm(T)*. The most basic correctness requirement is that any action tree T created by our implementation have perm(T) serializable.

The style in which we have defined serializability is more fundamental than the way it is defined in "traditional" concurrency control theory. Unlike the earlier definitions, this definition is applicable to systems which use multiple versions, as well as systems that use single versions of data objects. The earlier definitions included implicit information about versions, and so have required extensions [7], [8] to handle algorithms such as Reed's. In order to obtain a cycle-free characterization for serializability, however, it is necessary to put some information about versions back in. Thus, we define a structure similar to an action tree, but with a little more information.

Augmented Action Trees: In order to prove that our particular algorithm satisfies this basic correctness requirement, it is helpful to introduce additional structure into action trees, obtaining *augmented action trees* (AAT's). AAT's describe exactly which version (i.e. result of a particular other access to the same object) is read by each access. The definition of serializability is easily extended to AAT's, so that it is obvious that AAT serializability implies ordinary action tree serializability.

We prove a characterization for AAT serializability, in terms of absence of cycles in an appropriate dependency relation on actions.

Operations on Action trees and AAT's: The basic strategy we would like to follow is to define a set of operations on action trees which can be used to generate action trees. The actual strategy used is a little more complicated, however. Some of the correctness conditions we want to require involve relationships between "input requests" (originating from user programs) and "output steps" performed by the algorithm. (For example, our algorithm should only create a particular action A if the user program has actually requested the creation of A.) A convenient way to handle such conditions uniformly with invariants on action trees is to augment the global state information -- instead of a global state which consists of an action tree only, we let it be a pair (S, T) consisting of a summary of requests and an action tree. Then the operations we define will be operations on such pairs (S, T).

There are seven kinds of operations on pairs (S, T). Three "input operations" model requests by the user programs -- to create actions, and to commit and abort non-access actions. Four "output steps" model steps of the algorithm -- to create, commit and abort non-access actions and to perform accesses. Each operation has a set of conditions under which it is defined on (S, T), and a particular effect on (S, T).

A sequence of operations is *correct* provided either (1) some input operation is undefined on the result of its prefix, or else (2) *all* operations (both input and output) are defined, and for any (S, T) created during the execution of the sequence, perm(T) is serializable.

A similar set of definitions is given for AAT's. Interestingly, for AAT's, it is the case that the last condition (the serializability of perm(T)) actually follows from the definability conditions.

Execution Model: We wish to show that particular algorithms "generate" only correct operation sequences. In order to do this, we must define an execution model for distributed algorithms, and describe the sense in which an instance of the model generates operation sequences.

The model we define is a very simple automata-theoretic model, in which local computation steps are modeled by local state transitions, while communication is modeled by remote state transitions. We assume a homomorphism from automaton steps to operations on action trees (or AAT's). In this way, an execution sequence for the automata maps to an operation sequence for action trees (or AAT's).

Correctness Proof for Moss' Algorithm: A slightly simplified version (in which reads and writes are not distinguished) of Moss' algorithm [9] has been described using the given execution model, and a correctness proof carried out. The correctness proof has quite an interesting structure:

We imagine that the various nodes of the system implementing the algorithm are cooperating to create a virtual "global state," which is a request summary, AAT pair. The major work of the proof is showing that, provided the input operations are always defined on this global state, the output operations are also always defined.

A helpful device is to define a "possibilities" mapping from node states to sets of request summary, AAT pairs. This mapping describes the possible pairs which the node thinks could comprise the global state. We show that at all times during the execution, the true (virtual) (S, T) is in each node's set of possibilities. We also show that each node always performs operations that are defined for all (S, T) in its set of possibilities. Therefore, the required definability conditions are satisfied.

4.3. Extensions

We are currently extending the treatment in this paper to Moss' complete algorithm, which achieves extra concurrency by distinguishing "read" and "write" accesses and processing them differently. It appears that a rather general approach to modeling data objects and their accesses, including the case we have already treated and the read-write case as special cases, will turn out to be appropriate.

Our framework ought to be suitable for proving correctness of other implementations of nested transactions, such as Reed's [6]. We plan to study the applicability of our framework to Reed's algorithms.

The definitions we have developed so far just express the simplest correctness requirements, not subtle conditions such as correctness of orphans' views. We also do not address issues of fairness and eventual progress. We hope that the framework presented here will extend to allow expression of these other properties, and to allow correctness proofs for the difficult algorithms which guarantee these properties.

References

1. Liskov, B. H., Snyder, L. A., Atkinson, R. R. and Schaffert, J. C. "Abstraction Mechanisms in CLU," *Communications of the ACM 20*, 8 (August 1977), 564-576.

2. Liskov, B. and Scheifler, R. W. "Guardians and Actions: Linguistic Support for Robust, Distributed Programs," *Proceedings of the 9th Annual ACM Symposium on Principles of Programming Languages*, January 1982, 7-19.

3. Lampson, B. and Sturgis, H. "Crash Recovery in a Distributed Data Storage System," Xerox PARC, Palo Alto, CA, April 1979.

4. Eswaren, K. P., Gray, J. N., Lorie, R. A. and Traiger, I. L. "The notions of consistency and predicate locks in a database system," *Communications of the ACM 19*, 11 (November 1976).

5. Lamport, L. "Time, Clocks and the Ordering of Events in a Distributed System," *Communications of the ACM 21*, 7 (July 1978).

6. Reed, D. P. "Naming and Synchronization in a Decentralized Computer System," MIT/LCS/TR-205, MIT Laboratory for Computer Science, Cambridge, MA, 1978.

7. Kanellakis, P. and Papadimitriou, C. "On Concurrency Control by Multiple Versions," *Proceedings of the ACM Symposium on Principles of Database Systems*, March 1982, 76-82.

8. Bernstein P. and Goodman N. "Concurrency Control Algorithms for Multiversion Database Systems," *1982 ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1982, 209-215.

9. Moss, J.E.B. "Nested Transactions: An Approach to Reliable Distributed Computing," MIT/LCS/TR-260, MIT Laboratory for Computer Science, Cambridge, MA, 1981.

Publications

1. Burns, J., Jackson, P., Lynch, N., Fischer, M. and Peterson, G. "Data Requirements for Implementation of N-Process Mutual Exclusion Using a Single Shared Variable," *Journal of the ACM*, January 1982, 183-205.

2. DeMillo, R., Lynch, N. and Merritt, M. "Cryptographic Protocols," *Proceedings of the 14th ACM Symposium on Theory of Computing*, 1981, 383-400.

3. Fischer, M. and Lynch, N. "A Lower Bound for the Time to Reach Interactive Consistency," *Information Processing Letters 14*, 4 (June 1982), 183-186.

4. Herlihy, M. P. and Liskov, B. H. "A Value Transmission Method for Abstract Data Types," Computation Structures Group Memo 200-1, MIT Laboratory for Computer Science, Cambridge, MA, August 1981. To appear in *ACM Trans. on Programming Languages and Systems*.

5. Liskov, B. H. "On Linguistic Support for Distributed Programs," *Proceedings of the IEEE Symposium on Reliability in Distributed Software and Database Systems*, July 1981, 53-60. Also published in *IEEE Trans. on Software Engineering, SE-8*, 3 (May 1982).

6. Liskov, B, H. "Report on the Workshop on Fundamental Issues in Distributed Computing," *ACM Operating Systems Review 15*, 3 (July 1981), 9-38. Also published in *ACM SIGPLAN Notices 16*, 10 (October 1981), 20-49.

7. Liskov, B. H., and Scheifler, R. W. "Guardians and Actions: Linguistic Support for Robust, Distributed Programs," *Proceedings of the Ninth Annual ACM Symposium on Principles of Programming Languages*, January 1982, 7-19. Also Computation Structures Group Memo 210, MIT Laboratory of Computer Science, Cambridge, MA, November 1981.

8. Lynch, N. "Upper Bounds for Static Resource Allocation in a Distributed System," *Computer and System Sciences 23*, 2 (October 1981) 254-278.

9. Lynch, N. "Accessibility of Values as a Determinant of Relative Complexity of Algebras," *Computer and System Sciences 24*, 1 (February 1982), 101-113.

10. Lynch, N. "Multilevel Atomicity," *Proceedings of the ACM Symposium on Principles of Database Systems*, March 1982, 63-69.

11. Fischer, M., Griffeth, N. and Lynch, N. "Global States of a Distributed System," *IEEE Trans. on Software Engineering SE-8*, 3 (May 1982), 198-202.

12. Fischer, M., Griffeth, N., Guibas, L. and Lynch, N. "Probabilistic Analysis of a Network Resource Allocation Algorithm," *AMS Workshop on Probabilistic Algorithms*, June 1982.

Theses Completed

1. Berger, W. "Extension of a Real-time Display Editor," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

2. DuLong, M. "Formatter for Ada Source Programs," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

3. Schaffert, J. C. "The Specification and Proof of Data Abstractions in Object-oriented Languages," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, September 1981.

Theses in Progress

1. Bloom, T. "Dynamic Module Replacement in a Distributed System." Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1982.

2. Stark, E. W. "Foundations of a Theory of Specification for Distributed Systems," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected January 1983.

3. Henderson, C. "Locating Migratory Objects in an Internet," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected August 1982.

Talks

1. Liskov, B. "On Linguistic Support for Distributed Computing,"

    IEEE Symposium on Reliability in Distributed Software and Database Systems, Pittsburgh, PA, July 20, 1981
    University of Washington, Seattle, WA, August 1981.

Invited Lecture at the Inaugural Symposium, Cross Currents in Computer Science, University of New Hampshire, Durham, NH, October 23, 1981
Quality Software Meeting, Rockport, MA, October 30, 1981
General Electric Co., Schenectady, NY, November 10, 1981
Wang Institute, Tyngsboro, MA, December 1981.

2. Liskov, B. "Argus: A Language for Distributed Programs."

   Boston University, Boston, MA, February 10, 1982.
   Harvard University, Cambridge, MA, March 11, 1982.
   Prime Computer, Framingham, MA, April 11, 1982.
   Yale University, New Haven, CT, April 29, 1982.

3. Liskov, B. "Guardians and Actions: Linguistic Support for Robust, Distributed Programs," Ninth Annual ACM Symposium on Principles of Programming Languages, Albuquerque, NM, January 1982.

4. Lynch, N. "Towards a Theory of Complexity for Distributed Systems,"

   MIT, Cambridge, MA, Fall 1981
   Harvard University, Cambridge, MA, Fall 1981
   Yale University, New Haven, CT, Fall 1981

PROGRAMMING TECHNOLOGY

Research Staff

S.T. Berlin	J.C.R. Licklider
M. Blank, M.D.	S. Pinter
S.W. Galley	C.L. Reeve
L. Hawkinson	R.Sangal
P.D. Lebling	A. Vezza, Group Leader

GRADUATE STUDENTS

B.T. Berkowitz	T.F. Michalek
D. Lee	S.I. Ross
P.C. Lim	A. Yeh

Undergraduate Students

S. Barber	S. Malone
T. Bollinger	S. Nandapurkar
D. Brackman	R. Osgood
D. Dufour	J. Pezaris
S. Ferguson	R. Rotman
K. Hartman	M. Terpin
T. Kim	M. Vermeulen
	R. Wikrama

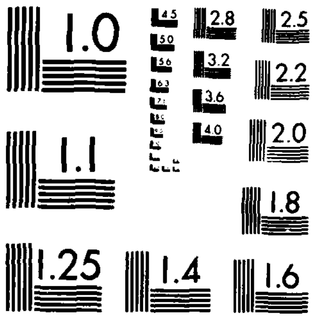
Support Staff

N. Mims	D. Venckus

AD-A143 459 LABORATORY FOR COMPUTER SCIENCE PROGRESS REPORT 19 1 JULY 1981-30 JUNE 19..(U) MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE.. M L DERTOUZOS 3/3

UNCLASSIFIED 01 MAY 84 MIT/LCS-PR-19 N00014-75-C-0661 F/G 9/2 NL

END
FILMED
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

1. INTRODUCTION TO PLANNING SYSTEM

The major activity of the Programming Technology Group this past year has been concerned with further development of the Planning System. The development activity has been focused on four major areas: (1) The multi-media aspect of the Advanced Message System (AMS); (2) a two dimensional user interface to the planning system; (3) PLAID in the areas of knowledge representation; planning graphs and reasoning capabilities; and (4) the machine independent MDL (MIM) system and environment including a VAX version of MIM which should achieve initial operational status within about a month.

2. ADVANCED MESSAGE SYSTEM

There have been four main areas of development in the Advanced Message System over the last year. First has been upgrading the Reader and Composer of the message system to handle multi-media (Lebling). Second was the development of a FAX decoding and display program in MDL (Galley). Third was the implementation of an encoder and decoder to transform multi-media messages into byte streams (Hartman). Fourth has been the implementation of a generalized two dimensional work space simulator and interface to the planning system. (Lebling).

2.1. Reader and Composer

The Reader and Composer can now deal with multi-media text as defined by ARPAnet protocols [1][2][3] Elements consisting of media other than normal text are logically inserted into the message by the user typing (for example) "Insert Fax <PDL>FOO.FAX". Work is proceeding on actually displaying these alternate media on our recently acquired BBN BitGraph terminal. At present, on non-graphic terminals (VT100s, for example), the media are not displayed. Instead a token is placed in the text field which indicates the presence of non-text media. This token may be edited as though it were part of the text the user typed.

When a message is received in multi-media form, the analogous process is carried out to display it; the Reader gives the user as much information as it can (usually just the medium and the origin of the "unpresentable" medium). The presentation is modular, so that when multi-media capable terminals appear, it will be possible to upgrade the presentation with little trouble.

2.2. Facsimile Images

Programs in MIM (machine-independent MDL) to decode, scale, and display facsimile images on the Apollo DOMAIN work-station were constructed.

The programs can decode images encoded according to the standards defined by Arpanet protocols [4][5][6]. These standards derive from the Dacom 450 and from the CCITT Draft Recommendation T.4 (Group 3) facsimile apparatus for document transmission. The standard specifies a two-dimensional run-length coding algorithm. Timing trials showed that FAXMIX in MIM was able to be decoded using the 2060 in 115 seconds. This is comparable to another implementation of a decoding program written in another language.[5] Because both the DOMAIN physical display surface and a virtual screen in the DIGRAM graphics system (under development by Lim) have only 1024 pixels from top to bottom, while the standard height of a facsimile image is some 2200 pixels, programs to shrink decoded images by half were constructed.

Currently the decoding program expects a file specification as its input, further work will focus on decoding and displaying FAX images contained in messages rather than files, displaying images for a user more easily and naturally, and encoding synthetic images with the CCITT method.

2.3. Dynamic Recursive Work Sheet Simulator

A generalized worksheet simulator has been written which builds on the ground cleared by VisiCalc. The goal has been to produce a multi-sheet, multi-user, data-based planning aid. To this end, query facilities have been implemented, including the ability to use worksheets as databases. Additionally, some of the recognized deficiencies of the commercially available VisiCalc have been remedied.

The query facility is based on two relatively simple additions to the basic work sheet simulator. First, the ability to manipulate references to locations in the worksheet. Second, the ability to have a function analogous to "for-each" or MDL MAPF. An example may help clarify these additions.

Suppose a worksheet contains a database of products in a certain area. Successive rows in the worksheet would be different suppliers for the product, and each column would have a meaning. For example, a row might consist of a supplier, a price, and how long delivery of an order would take. The user would set up the selection criteria for a particular order:

```
AND(PRICE<100,DELAY<10)
```

This expression would determine whether some particular entry fits the selection criteria, but it is not general in that it points to absolute locations within a worksheet (i.e, PRICE and DELAY). However, if one can get a pointer to a worksheet entry into some other entry (say, E), one can reference locations relative to it:

[5]A.R. Katz's implementation in L10.

```
AND(PRICE(E)<100,DELAY(E)<10)
```

Assuming PRICE and DELAY are appropriate offsets into a record, the user can test a record by pointing to it from E. The only other facility needed is the MAPF that lets successive records be examined:

```
FIND(AND(PRICE(*)<100,DELAY(*)<10),SUPPLIERS(A1...A100))
```

Where the special symbol * means "the record currently being examined." If a record matches the test, a pointer to it is returned. By making the requirements (maximum price and maximum delay) variables, a forms based query system comes into existence.

```
FIND(AND(PRICE(*)<PMAX,DELAY(*)<DMAX),SUPPLIERS(A1...A100))
```

The user might see only Supplier Frobozz Electric Maximum Price 100.00 Price 95.95 Maximum Delay 10 days Delay 7 days

By changing the price or delay criteria, the user would see which suppliers are acceptable. Commands exist to step through the records which match a given search specification.

Some of the additional capabilities which have been implemented include:

- The ability to have any number of worksheets loaded at once. In addition, different worksheets may be displayed in each window. Changes in one worksheet which impact others cause automatic updating just as though only one worksheet existed.

- The ability to reference entries in different worksheets than the current one. In the example above, "SUPPLIERS" is a separate worksheet from the one in which the querying is being done. Other worksheets will be loaded dynamically as needed if they are referenced by a worksheet already loaded.

- String comparison operations, including substring operations, prefix matching, and exact matching.

- The ability to define new operators. These will be dynamically loaded as needed.

- "Area" operations (as opposed to "entry" and "range" operations). For example, the "Print" command (used for getting hard copy of a worksheet) will take an arbitrary sub-part of the worksheet as one of its arguments.

- Form definition and filling commands. A group of entries may be defined as a form, after which the user may use the "Fill" command to step through those entries sequentially.

Many of the deficiencies of the commercially existing VisiCalc have been removed.

- The command parser determines heuristically whether a given line of input is a command (such as "global format integer"), a string entry (such as "Maximum Price") or an expression (such as "PRICE + 100").

- The user is completely free to use forward and circular references. This means that the layout constraints imposed in the basic VisiCalc (that all references must be either to previous rows or previous columns) are removed. The user is therefore free to organize the worksheet in his own way.

- Any column may have its width set independently of the other columns. In VisiCalc all columns must have the same width.

- Any entry or range may be named, and these names may be used in place of the absolute entry coordinates at any time.

It is intended that the worksheets or parts thereof will be usable as media in the message system.

3. KNOWLEDGE-BASED PLANNING AIDS

The principal objective in this research area is the development of a high-level planning aid system, to be known as PLAID. (Hawkinson, Sangal, Ross, Michlich, Terpin, Yeh) The primary purpose of PLAID will be to assist planners in developing plans, but PLAID will also be useful in monitoring what it has helped plan. Monitoring aids in PLAID will include query, history/status report, and alert/reminder facilities.

PLAID encourages a structured style of planning by (1) requiring that every plan have one or more topics, (2) allowing a plan to be separately elaborated for distinct subtopics of its topics, and (3) allowing plans for distinct subtopics of a topic to be combined to form a new plan for that topic. Thus, for example, a plan for a project might be separately elaborated for each of several tasks, and then resulting plans for the tasks might be combined in various ways to form new elaborated plans for the project. PLAID supports concurrent exploration of alternatives by allowing a plan to be separately elaborated for each of several alternative (and typically conflicting) hypotheses. As might be expected, separate elaboration of plans for alternative hypotheses or for distinct subtopics is allowed at any point, and hence relationships between an original plan and elaborations of it can become arbitrarily complex. For example, a particular elaboration could be an alternative elaboration of a plan for a subtopic of some alternative elaboration of a plan for a subtopic of the original topic.

To enable a planner to visualize the present state of his planning, PLAID will be capable of displaying graphically the relationships between an original plan and various of its elaborations (each of which is itself a distinct plan). A planner may operate on such a *planning graph* by (1) selecting a particular plan (node) for further elaboration; (2) adding a hypothesis; (3) specifying subtopics of the topic of the currently selected plan, to allow separate elaboration of the plan for each subtopic; (4) specifying a set of alternative hypotheses, to allow separate elaboration of the currently selected plan for each of these alternative hypotheses; (5) specifying a set of plans to be combined; (6) discarding plans that are no longer of interest; (7) selecting the best among a set of alternatives (implicitly discarding the rest); etc.

In addition to providing operations on planning graphs, PLAID will assist planners by (1) evaluating plans upon request, (2) detecting "obvious inconsistencies" (often constraint violations) in plans and helping resolve them, (3) presenting work sheet views of plans and allowing indirect amendment of plans via such work sheets, and (4) producing documents based on plans. A plan evaluator now being developed in a Master's thesis (Ross) attempts to determine a plan's resource requirements and likelihood of success. Resolution of a detected inconsistency in a plan involves the revision or retraction of at least one hypothesis or belief (possibly even a constraint or rule); the revision or retraction of a hypothesis or belief necessitates reconsideration, and usually revision or retraction, of all hypotheses and beliefs it supports (inferentially). work sheets are a highly popular planning device, especially as implemented on display terminals. The coupling of worksheets to plans in planning graphs provides a very powerful tool for planners, a tool which permits planners to view plans from various perspectives, to make projections based on plans, to compare alternative plans, to elaborate or revise plans indirectly by filling in or changing values on work sheets, to maintain consistency among multiple work sheets coupled to the same or related plans, to view the result of combining separately elaborated plans (even as they are being revised by several individuals, say), etc. Documents such as budget proposals can be automatically produced from plans in much the same way as work sheets are automatically filled in from plans, except that English paragraphs also have to be generated.

To make it practical to use PLAID in a new organization, PLAID must be able to acquire and update whatever organization and application specific information it needs from sources within the organization. Such information would include individual facts and beliefs, descriptions of the organization and of its record-keeping structures, program and project descriptions, procedures, rules and policies, constraints, event descriptions, and hypotheses for plans. Most such information would be acquired by model-directed interview (with communication via English phrases and displayed menus), but some might be acquired directly from databases otherwise maintained by the organization.

3.1. PREP Development

Our knowledge representation system, PREP, has been improved at both the design and implementation levels, especially in the areas of notation, concept types, and belief representation. Planning graphs have been further developed and implemented. Mark Terpin, an undergraduate member of our group, is using planning graphs in a system to aid MIT computer science students in planning their courses of study.

Various reasoning capabilities are being added to PREP, some of a general nature (e.g., inheritance, maintenance of viewpoints, and reasoning about time) and some specific to planning (e.g., decomposition and plan evaluation). The more general reasoning techniques are being developed in a Master's thesis (Michalek) entitled "A Rigorous Approach to Some Basic Inference Problems"; this thesis, which also presents a formal semantics for PREP, was completed in May of 1982. The techniques for plan evaluation will be the topic of a Master's thesis (Ross) tentatively entitled "A Plan Evaluation System"; completion of this thesis is expected by September 1982. Another member of our group (Yeh) is working to develop a Master's thesis in the area of plausible reasoning.

Our knowledge representation methodology has been refined, particularly in the area of actions and processes. Our sample knowledge base, which describes a particular organization and its budget planning process, has been revised in accordance with our refined methodology and models.

4. MIM DEVELOPMENT

Progress in the machine independent MDL (Reeve, Blank, Berkowitz, Lim, Brackman, Berlin, Dufour, Hartman) project over the year has been in the areas of: (1) Implementation of MIM for Apollo Domain; (2) Garbage collections/storage management; (3) Flexible new stream I/O system; (4) MIM Graphics (DIGRAM); (5) Porting parts of the MDL environment; (6) Working on the VAX implementation; (7) Adding further optimizations to the compilers; (8) Increasing the robustness of the various implementations.

4.1. MIM for the Apollo Domain

An open compiler that translates MIM virtual machine instructions to M68000 instructions was developed. The code produced by the open compiler in conjunction with a kernel, implemented in assembly language for the Apollo Domain provided the necessary scaffolding and foundation for bootstrapping the entire MIM system onto the Apollo. This version of MIM was used as the basis for the design and implementation of the DIGRAM Graphics System.

4.2. Garbage collection/Storage management.

Both a mark/sweep and full-copy garbage collector have been implemented for MIM. They were both written in MDL and both work well. Both garbage collectors run on the Apollo and TOPS-20. In addition, a flexible "zoned" storage system has been implemented to permit both garbage collected and non-garbage collected storage to co-exist.

4.3. Stream I/O System

A completely new implementation of I/O is underway for MIM. An initial version is up and running on the TOPS-20 version of MIM. Porting the I/O system will be somewhat more difficult than other subsystems since I/O is inherently more machine dependent that most programs.

4.4. MIM Graphics

MIM Graphics is provided by the DIGRAM system. Three aspects of it have been under development: (1) Virtual graphics device (DIGS); (2) Graphics Run-Time Support System (GRSS); (3) A Test application program.

The DIGS module provides basic graphics operations for programs written in MDL in a uniform manner. The DIGS module provides graphic operations in Motorola M68000 assembly language not provided by the Apollo Domain hardware. In particular, that module provides the raster operations (also known as bit-blt) because the Apollo Personal Computer does not. DIGS also provides the triangle filling operations for filling polygon areas. DIGS provides the primitive operations of DIGRAM.

GRSS provides graphics application programs with graphics input functions, graphics output functions, and viewport manager functions. The graphics input functions obtain input from graphics input devices. The graphics output functions display lines and triangles, combine boxes with raster operations, and print text on a viewport. The viewport manager functions manage those viewports, or virtual screens, by providing functions to create, modify, and destroy viewports. We have written those functions in MDL, compiled them on MIT-XX computer, moved them to the Apollo Personal Computer, and used those functions on the Apollo.

Finally, we have built a graphics application program to test the graphics system. That program is a simple intelligent screen simulator which allows MDL to read input from the keyboard and print output on the screen. In addition to the basic display operations such as erasing a character for a delete operation, that intelligent screen simulator uses the viewports in the graphics system to restrict the output to specific areas of the screen. Since developing graphics programs is much easier when we can see the program and the graphics output simultaneously, that intelligent screen simulator will make the development of graphics programs quicker.

Most of the implementation mentioned above has been accomplished and a Masters Thesis (Lim) describing the DIGRAM system in more detail was completed in May.

4.5. Porting of the MDL environment to MIM.

The following subsystems have been moved from MDL to MIM: (1) EDIT-the MDL structure editor; (2) PRINT-the MDL pretty printer; (3) &-Printer-the complex structure printer; (4) PACKAGE-the MDL package system; (5) FINDATOM-identifier pattern matcher; (6) INT-the MDL interrupt system.

Thus approximately 75% of the MDL environment has been moved to MIM. The MIM system is currently almost as powerful a program development environment currently found in the MDL system. It is expected that by early next year the MIM environment will achieve parity with the current MDL environment.

4.6. VAX Implementation.

An open-compiler and a MIM kernel are currently under development for the VAX that was recently acquired. This implementation will run on the "personal" VAXes that LCS has acquired. MIM will be up on the VAX in about a month.

4.7. Compiler Optimizations

As weaknesses in the code produced by the MIM compilers were encountered, optimizations were included to correct these weaknesses. Optimizations were added to both the MIMC compiler which compiles MDL into MIM and the various MIMOC compilers which translates MIM into order code for the target machines.

4.8. Robustness Enhancements

MIM has been acquiring some real users, inevitably each new real user uncovers new bugs in the MIM. As the bugs have been addressed, the overall reliability of MIM has increased. Our goal is to have MIM achieve the robustness of MDL by the end of the year.

References

1. Postel, J. B. "Internet Message Protocol" Arpanet Network Information Center, Reprot RFC-759 August 1980.

2. Postel, J.B. "A Structured Format for Transmission of Multi-Media Documents" Arpanet Network Information Center, Report RFC-767 August 1980.

3. Postel, J.B. "Rapicom 450 Facsimile File Format" Arpanet Network Information Center, Report RFC-769 September 1980.

4. Katz, A. "Decoding Facsimile Data from the Rapicom 450" Arpanet Network Information Center, Report RFC-798 September 1981.

5. Agarwal, A., O'Connor, M.J., and Mills, D.L. "Dacom in 50/500 Facsimile Data Transcoding," Arpanet Network Information Center, Report RFC-803 November 1981.

6. CCITT "Draft Recommendation T.4-Standardization of Group 3 Facsimile" Arpanet Network Information Center, Report RFC-804 undated.

Publications

1. Dornbrook, M., Blank, M., "The MDL Programming Language Primer" MIT Laboratory for Computer Science, Cambridge, MA, 1980.

2. Licklider, J.C.R. "National Goals for Computer Literacy" in Book, Computer Literacy, 1982, New York, 281-287.

3. Licklider, J.C.R. "Teleconferencing: Working Together Across Space and Time" in Book Innovations in Telecommunications (Part B), 1982, New York, 949-993.

4. Licklider, J.C.R. and Vezza, A. "The Utility of Electronic Message Systems" in Book ELECTRONIC MAIL AND MESSAGE SYSTEMS: Technical and Policy Perspectives, Arlington, VA, 11-31, 1981.

5. Vezza, A. (ed.) "ELECTRONIC MAIL AND MESSAGE SYSTEMS: Technical and Policy Perspectives," American Federation of Information Processing Societies, Inc. 1981.

Theses Completed

1. Lim, P.C. "A Device-Independent Graphics Manager for MDL," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1982.

2. Wallace, K.G. "Canonical Terminal Support in the TOPS-20 Operating System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

3. Craig, T. "An Analytic Model for the Performance Evaluation of RMDS: A Relational Database Management System, " S.M. thesis, Department of Electrical Engineering and Computer Sci[illegible]e, Cambridge, MA, June 1982.

Talks

1. Licklider, J.C.R. "Computer Literacy," Interdisciplinary Conference on the Future of Literacy, University of Maryland, Baltimore County, MD, April 1982.

2. Licklider, J.C.R. "The Electronic School," Conference on the Future of Electronic Learning, Columbia University, New York, NY, April 1982.

3. Licklider, J.C.R. "Some Ideas About a Graphical Programming and Program Monitoring Language," ARPA Conference on Graphical Representation of Software, Key West, FL, December 11-14, 1981.

4. Licklider, J.C.R. "Education in the 1990s," Keynote Address at Fourth Annual Computer Conference, Lesley College, Cambridge, MA, May 8, 1982.

REAL TIME SYSTEMS

Academic Staff

M.L. Dertouzos
R.H. Halstead
S.A. Ward, Group Leader
R.E. Zippel

Research Staff

J. Arnold
S. Keohan
J. Test

Graduate Students

H. Alcabes
T. Anderson
D. Esposito
D. Goddeau
M. Johnson
A. Lake
J. Mandry
R. McLellan
A. Mok
J. Powell
D. Rubine
L. Seiler
R. Simmons
T. Sterling
T. Teixeira
C. Terman
J. Triosi
D. Vogel

Undergraduate Students

D. Alpern
S. Goldman
B. Hankins
D. Hills
K. Holmes
W. Hughes
J. Kesselman
G. Kramer
J. Krueger
C. Li
J. Loaiza
T. Lukac
J. Morrison
M. Nahabedian
G. Pratt
N. Ronkin
M. Rose
E. Seidman

J. Lakos	R. Spellman
C. Le	T. Tsakiris

Visitors

G.C. Clark

Support Staff

M. Brown	R. Kane
O. Feingold	E. Tervo
J. Hoppe	P. Vancini
L. Kenen	

1. INTRODUCTION

Major projects within the Real Time Syst ms Group during the past year have been (i) continuing development and technology transfer of the NU personal computer system; (ii) continued study of multiprocessor architectures, leading to a new effort directed at the design and construction of a substantial facility for experimentation with multiprocessor systems; (iii) substantial redesign of the TRIX network-oriented operating system, along with initial stages of its corresponding reimplementation; and (iv) continued work in the area of VLSI design, with particular emphasis on the development of design tools.

2. THE NU PERSONAL COMPUTER

During the past year, work on the Nu project has been largely in two areas: (i) hardware redesign and other aspects of technology transfer, in cooperation with Western Digital; and (ii) development and enhancement of software, particularly the microprocessor-based UNIX system which is currently used on the Nu.

Specific accomplishments include:

1) In cooperation with Western Digital, a substantial redesign of the Nu implementation. Changes are directed primarily at performance improvements over the existing prototypes, and include

   a) Additional bus protocols for the efficient transfer of multi-word packets. This extension improves the NuBus bandwidth to a maximum of about 35 MBytes/second.

   b) Incorporation of a System Diagnostic Unit, which enhances reliability and maintainability by independently verifying the integrity of all major system components.

   c) Substantial changes to the M68000-based CPU card, including addition of local cache and revised memory mapping.

2) Development of a display-multiplexing window system, and its integration into UNIX (Test). The windows each provide the full functionality of a virtual terminal, and may be reconfigured dynamically under user control.

3) Continued improvements to, and maintenance of, our microprocessor-based UNIX and C systems (Terman, Test).

As the Nu project enters its fifth year, several remarks on its retrospective evaluation seem appropriate. The project's goals have been (i) tool building: to provide the laboratory with a desperately needed resource; and (ii) technical: to explore and exploit a number of architectural ideas and technology, ultimately in order to influence the state of the art.

Progress toward the first of these goals has been disappointing. Production Nu's are currently scheduled for early 1983, and we continue to anticipate their widespread assimilation by LCS; however, delays in their availability have largely eclipsed their critical role as laboratory resources. As the project began, there were simply no commercially available personal computers which satisfied the Laboratory's projected needs; indeed, this void was the major impetus for the Nu effort. Over the intervening years, plausible commercial alternatives have arisen, partly as the result of the Nu and similar projects elsewhere. During this period, LCS expended considerable manpower and frustration negotiating a series of manufacturing arrangements (Exxon, Heath, Zenith, and Western Digital); while we expect the Western Digital connection to prove fruitful, it seems clear that our approach to technology transfer has not been an unqualified success.

By technical standards, however, the project has been both successful and influential. The LCS prototypes became operational quickly (3 months for the initial 8086-based Nu, less than a year for the NuBus-based system) and have reflected positively on the underlying architectural ideas. The Nu's architects have devoted substantial energy to interactions with manufacturers of similar machines, which evidence considerable influence from the structure and goals of the Nu. System software used on the Nu, most particularly our revisions of Western Electric's UNIX software for microprocessor (8086, Z8000, M68000, and N16000) environments have been widely circulated and are something of a standard on microprocessor-based UNIX systems. We have repeatedly had the experience of discovering that the microprocessor-based UNIX system we are being introduced to is running software developed at RTS.

It is noteworthy that the Nu's architectural flexibility is attracting commercial attention. Lisp Machines, Inc. is developing a NuBus-based LISP processor to be available for the Western Digital machines. Western Digital, in turn, has plans to offer a UNIX system which supports multiprocessor Nu systems.

3. MULTIPROCESSOR ARCHITECTURES

The period since June 1981 has seen continued development of the MuNet multiprocessor project and MuLisp parallel programming language, as well as the initiation of major new projects to build a multiprocessor experimentation facility and a real-time interactive graphics system.

3.1. MuNet and MuLisp

Accomplishments during the past year include:

1) Development of an acceptably efficient simulator for the LCODE "machine language" of the MuNet; along with an efficient translator from MuLisp to LCODE (Halstead).

2) Augmentation of MuLisp and LCODE to handle "futures" as a convention for argument passing (Ronkin).

3) Implementation of several test programs in MuLisp, and the collection of statistics about the achievable parallelism in their execution. Notable among these programs are part of a graphics system (Simmons) and a parallel alpha-beta search using the "mandatory work first" strategy (Hankins).

The results obtained from these simulations are generally encouraging, but there is a real need for a facility capable of handling larger programs with longer execution times, for it is these programs that have the greatest need for parallel processing. This need is a principal motivation for the multiprocessor experimentation facility, discussed next. Long-term continuation of the MuNet research will involve the development of *substantial parallel programs* using the multiprocessor experimentation facility. The observed properties of these programs will then guide the specification of a MuNet architecture, and the programs themselves can serve as benchmarks against which proposed architectures can be judged.

3.2. A Multiprocessor Experimentation Facility

In order to involve large numbers of people in the development of parallel programs, and thus amass a body of expertise concerning parallel programming, it is imperative to have an execution environment which rewards (with faster execution) the construction of programs in a parallel manner. Researchers deeply committed to parallel architectures will be willing to simulate parallel execution on a sequential machine, and derive satisfaction from the timing numbers printed by the simulator, but rapid progress in parallel architectures requires that a wider user community be tapped. Furthermore, even researchers in the field will be reluctant to experiment with programs whose simulation on a sequential machine is too time-consuming, yet many of these larger-scale programs are precisely the ones that should receive more attention. Finally, innovation often comes from unlikely sources, and the more individuals of every persuasion are involved with parallel processing, the more chance there is for innovation to occur.

A major difficulty facing anyone who would like to experiment with a parallel implementation of some algorithm is the lack of any existing infrastructure for executing programs on parallel machines. The prospect of man-years of tool-building will dissuade all but the most dedicated advocate of parallel computing. We believe this effect has operated in many instances within the Laboratory for Computer Science, as well as outside.

Most candidate applications for parallel machines do not require highly specialized architectures in order to benefit from parallel execution. Mostly, stable and robust hardware is needed, along with a user-friendly program development system. Almost always, even algorithms targeted for specialized hardware can profitably be tested, debugged, and refined using a reasonably competent general-purpose machine.

Accordingly, we propose to build a multiprocessor experimentation facility, to be generally available to researchers at MIT who are interested in parallel computation. We see two principal categories of users:

1) Serious developers of parallel architectures and languages who would like to quickly breadboard their ideas prior to full implementation.

2) Workers in other disciplines who would like to investigate parallel processing as a solution to their problems but cannot afford to construct their own multiprocessor system.

Hardware and software design for the proposed multiprocessor experimentation facility, whose working title is SPUDS, has been conducted during the past year (by Zippel, Halstead, Anderson, Sterling, Alpern, Mercado, and Morrison). The broad outline of the hardware design is firm; software design for a general-purpose user-friendly programming environment has begun and is continuing.

Hardware Design: The principal hardware design goal is maximum use of commercially available components, consistent with meeting the performance and other objectives of the system. We have located a commercial source (Microbar Systems) of unique processor and memory boards that are especially suitable for our system. Each processor and memory card is dual-ported: one port connects to a standard Intel Multibus (using four extra lines to allow 24-bit addresses), and the other port connects to a special Microbar "fast bus." This organization makes it possible to attach several processors to one Multibus by pairing each processor with one or more memory cards to which it is directly connected by its own fast bus. Any processor can still access any memory location in the system, but accesses to its own set of memory cards are performed on its own fast bus and do not load the Multibus. An incidental bonus is that accesses over the fast bus complete in 500 ns, allowing 8MHz microprocessors to run at full speed.

One might propose constructing a large multiprocessor simply by connecting a large number of such processor-memory pairs to a single Multibus. The fast-bus transactions do allow one Multibus to serve more customers, but beyond a certain number of processors, Multibus saturation is likely. There are also logical limits, relating to the interrupt structure of the Multibus, that make it difficult to attach more than eight processors to one Multibus. Therefore, some other strategy must be used to build larger multiprocessors.

The RingBus: We plan to construct a multiprocessor out of "clusters," each of which is a Multibus backplane containing approximately four processor-memory pairs as described above. Multi-cluster systems will be bound together by a special segmented-bus structure (the "RingBus") that will function as a memory with up to eight ports in which every memory location is accessible from every port. Physically, the RingBus is packaged as a number of "sectors," each of which is a board that plugs into a Multibus, behaves like memory from the point of view of devices on that Multibus, and is connected to other sectors. (Each sector also contains some control registers.) The RingBus also contains one central arbiter board, which is connected to all the sectors and controls the granting of requests to use the RingBus. RingBus design has been performed primarily by Anderson, Halstead, and Sterling.

The RingBus is a segmented bus in the shape of a ring, with each sector corresponding to one segment of the bus. Switches between the segments are controlled by the arbiter, and may be closed to fuse several segments together for a particular transaction, or opened to allow segments to operate independently. Thus, if each cluster is performing accesses to memory in its own sector of the RingBus, all such accesses can proceed concurrently. For a cluster to access memory in some other sector, some number of segments must be assigned to form a connection between the requesting cluster and the requested memory. The RingBus design is thus optimized for local accesses, but is capable of performing nonlocal accesses without great penalty. Several nonlocal accesses may even proceed concurrently, provided they use non-overlapping sections of the RingBus.

The arbiter's job is to choose between requests that require conflicting sections of the RingBus. With the limitation of a maximum of eight sectors per RingBus, we believe that a synchronous arbiter can be constructed that will act on requests within 200-300 ns -- an acceptable delay for global memory requests.

Expansion of the System: A full-size (eight-sector) RingBus, connected to Multibuses with four processors each, will form a system with 32 processors, and this is our intermediate-term objective. However, we would like to be able to experiment with larger multiprocessors.

Redefining the RingBus to allow more ports entails increasing cost to maintain the same performance level, and there are practical limits on how many processors can be served by one Multibus. Although shared memory is an efficient communication technique for moderate numbers of present-day microprocessors, it does not scale well. Therefore, larger systems that we would propose do not have any shared memory that is accessible as ordinary memory to every processor in the system. Rather, we propose to build larger systems out of several rings (one RingBus and up to eight Multibuses per ring), using some commercially available point-to-point or network communication hardware (e.g., 10 MB Ethernet) to connect them. An advantage of the basic SPUDS architecture is that one or more processors can be dedicated to handle the communication protocol, if necessary, while remaining tightly coupled to the processors they serve. There is thus almost unlimited potential for constructing "smart peripherals" using only off-the-shelf hardware.

The proposed expansion via multiple rings does not appear to require any hardware work, but it must be kept in mind while designing the software environment, to avoid making decisions which would preclude making effective use of multiple-ring systems.

Metering: In addition to its shared memory and control functions, each RingBus sector will perform a number of monitoring functions, allowing observation of usage patterns both on the RingBus and on individual Multibuses. This will be generally useful in tuning the performance of programs, and will also enable researchers in architecture to measure the access patterns and other performance attributes of their programs. Specification of the metering functions has been undertaken by Sterling.

Hardware Plans: An operating prototype RingBus with two sectors is planned for September 1982. By July 1982, we should have two Multibus cages and five processor-memory pairs (we plan to use Motorola 68000 processors; Microbar also sells Intel 8086/8087 boards). By summer of 1983, we plan to expand to a full-ring 32-processor system (approximate power: 10 MIPS), at an approximate parts cost (including power supplies, connectors, etc.) of $5,000 per processor memory-pair.

Software Environment: The major goal of the software environment is to allow use of the system at a number of levels of sophistication, from a bare machine (with only downloading and configuration support being used) through several levels of utility subroutine support (e.g., at different levels different communication styles will be supported; at some levels "one program per processor" will be the rule, but other levels will support multiplexing a processor among a number of processes), to a high-level language with parallelism integrated into its semantics. The software tools will generally be structured so as to be accessible from the programming language C, which we expect to be the primary implementation vehicle.

At all but the highest (parallel programming language) level, a user will write one or more programs in C and specify a configuration in which they are to be downloaded. A configuration language will allow a desired configuration pattern to be specified without direct reference to specific processor names. This is being worked on by Mercado and Halstead.

A set of communication and memory management utility routines is being specified by Alpern. These routines will support message-based communication (may occur even between different rings) and allocation of memory shared among a designated set of processors (restricted to all be on the same ring).

A debugger is being specified by Morrison. Most of the debugger operations will be performed on a host machine connected to SPUDS, but a basic set of "probe" primitives on SPUDS itself will allow the debugger to perform its operations.

Zippel and Halstead have begun to design a multiprocessor Lisp (working title: MultiLisp) to serve as an integrated high-level language tool for users of SPUDS. MultiLisp will be influenced by the constructs of MuLisp and also, probably, by SCHEME.

Specification will be completed, and implementation begun, on all of these software tools during summer of 1982.

Higher-Level Software Tools: Several higher-level software projects contemplate using SPUDS as a source of computer power. Considerable work has already been performed by Sterling on the specification of an event-based control flow programming language and architecture, where the normal SPUDS processors would act as slaves, directed to work on tasks by one or more specialized high-performance "dispatcher" processors.

Another likely project is a generalized emulator facility that will use multiprogramming to speed the simulation of computer architectures. A related application of interest to Zippel is the simulation of LSI integrated circuits.

3.3. Interactive Real-Time Graphics

In developing multiprocessor architectures and languages, we must be guided by experience with actual, substantial programs. A considerable amount of analysis of numerical programs has been conducted in connection with the various data flow computer architecture projects. Less attention has been given to coherent programming methodologies for real-time problems, or for symbolic data-structure-intensive programs (programs we would normally be inclined to write in a programming language such as Lisp). There is reason to believe that real-time and symbolic computation will benefit from different programming languages, and perhaps different arcnitectures, than are suitable for purely numeric computation.

The MuNet project, discussed above, is concerned with finding languages and architectures that are good for solving these problems. A problem that has both real-time and symbolic components (as well as an appreciable numerical component) is real-time, interactive, three-dimensional graphics. This has been chosen as an application to be implemented on SPUDS using MultiLisp technology. This will provide good calibration for the usefulness of MultiLisp and also provide a benchmark (the graphics program) for judging proposed MuNet architectures. Furthermore, it will attack a problem of considerable practical importance. It is likely that a general-purpose multiprocessor can provide a more scalable, cost-effective graphics system that the traditional approach of special purpose processors using extremely high speed hardware.

Why a General-Purpose Machine? Many graphics researchers advocate special-purpose VLSI hardware (e.g., the "Geometry Engine"). Special-purpose hardware can certainly perform specific functions faster than general-purpose hardware, but it is often difficult in special-purpose hardware to insert the flexibility to respond to information that may become available at various points during processing. For example, a geometry engine might take polygons in three dimensions, transform them into the viewer's coordinate frame, clip them to a window, perform a perspective transformation, and finally draw them. A general-purpose machine, however, would be able to detect, after a few transformations, that an entire object is behind the viewer, or obscured by another object, and thus save the work of processing the remaining polygons associated with the object. A general-purpose machine would also be able to adjust the amount of detail in an object according to its distance from the viewer. These techniques could easily save orders of magnitude of work, enough to more than compensate for the speed advantage of special-purpose hardware.

Special-purpose hardware does have a place in graphics processing, but ideally it would be as a simple speed enhancement for general-purpose operations (e.g., a floating-point multiplier), so that there are many convenient opportunities in the course of a computation for the flexibility and "intelligence" of general-purpose computing to come into play.

The Planned Graphics System

The planned graphics system will execute on a full SPUDS system of 32 processors or more. Output from the program will be in the form of a display list (more compact than a bit map) in the shared memory of the RingBus. The display list will indicate the left edge of every distinguishable region in the picture, along with the color of the region. Smooth shading between colors will be possible for better modeling of curved surfaces. A special piece of hardware (attached to some SPUDS cluster) will fetch the display list and generate a color picture "on the fly" in raster-scan order.

Current Status: A program (approximately 1500 lines of code) for displaying polygonal shapes in three dimensions has been developed by Halstead. The program produces the display list format discussed above. The display list can then be viewed on a bit-map color monitor (512 x 512 pixels). Programming wizardry by Goddeau has led to the generation of a number of interesting shapes to display. The current program is being used to test the image quality that results from various display approaches, preparatory to specifying display hardware design. Current results suggest that picture complexities of several hundred polygons are feasible.

4. TRIX OPERATING SYSTEM

TRIX is a kernel operating system designed for use on a network of interconnected processors; its general semantics and structure have been described in previous progress reports and elsewhere. During the past year, work by Halstead, Sieber, Test, and Ward of RTS and Dave Clark of the CSR group has led to a substantial revision in its implementation strategy; a reimplementation of the newly structured TRIX is in progress. We refer to the new implementation as TRIX 1.0, to distinguish it from the previous TRIX 0 implementation which became operational in early 1981.

Briefly, the semantics of TRIX revolve around a stream communication mechanism, rather than passive objects such as files or processes, allowing uniform and transparent access to local and remote objects. The basis of TRIX semantics on communication paths yields several interesting properties: (i) minimization of the kernel system (directories and naming mechanisms, for example, may be left to higher-level software); (ii) abstraction of function from implementation (a file containing the digits of PI, for example, may be indistinguishable from a process which computes them on demand); and (iii) a single, "anarchistic" name space in which interpretation of names and access to the services they denote is controlled by local rather than global mechanism.

The initial implementation, TRIX 0, based its communications on a pure, asynchronous message passing mechanism. A major defect in that implementation is its tendency to impose certain overhead costs associated with its very general message passing semantics on all programs, regardless of their requirements; thus simple programs which conform naturally to the more conventional procedure call semantics incurred an unwarranted performance overhead. TRIX 1.0 is thus motivated largely by our desire to confine the costs of asynchronous message-passing to those programs which require its power.

A key to the TRIX 1 implementation is the semantic distinction between the static environment of a process and its more ephemeral state. To this end, we shall refer to three interrelated kinds of resources supported by the system: domains, ports, and threads.

A domain comprises a set of addressable memory words at specific addresses and a set of handles for ports, accessible via specific handle IDs. The concept of a domain thus resembles the concept of an "address space," but note that a domain does not have its own program counter, stack pointer, fast registers, etc. These are the property of threads that may execute in the domain. The handles may be viewed as a generalization of open file descriptors; the set of handles accessible to a domain completely determine the capacity for communication between that domain and the external world.

A port is a means for communication from one domain to another. Every port has one specific domain that is its handler ; associated with every port are one or more handles possessed by domains (possibly including the handler domain). Control may be transferred from a domain that has a handle on a port to the handler domain of that port by REQUEST or RELAY system calls.

A thread is a single sequential path of execution under TRIX; threads are the units among which CPU time is divided and incorporate some of the elements of "processes" in other systems. Each thread has a private stack, but also borrows from the address space of whatever domain it is currently executing in. In general, several threads can execute concurrently in the same domain, and a domain may exist without any threads currently executing in it. A thread carries with it various useful attributes, e.g. , saved register contents, scheduling information, etc.

Threads in TRIX are created using the SPAWN system call. A thread T , in general, has a parent thread , which is the thread that executed the SPAWN operation that created T. A thread is said to be its parent's attached child . TRIX thus imposes a tree structure on threads. As an option to the SPAWN system call, a thread can also be created detached , which means that it has no parent thread.

The primary communication mechanism in TRIX is the message send or request mechanism, in which a thread executing in a domain D having a handle on port P moves into the domain D' which is the handler of port P. This process resembles the domain call mechanism on some other systems, and is supported by the REQUEST and RELAY system calls; return is effected via the REPLY system call. All three of these allow the communication of some small amount of information (plus, optionally, one port handle) directly with the call.

A call can also be accompanied by a data window giving the caller access to a segment of the caller's address space, allowing transfer of data between caller and caller domains by means of the FETCH and STORE system calls. Associated with the data window are the address of the segment in the calling domain, the size of the segment in bytes, and a "current position" offset in the segment. There is also an allowed I/O direction , which may permit both reading and writing, or restrict access to only one of these operations.

Unlike usual operating systems which provide for naming of files (and in some cases a restricted set of other objects). TRIX is oriented toward the naming of handles. In many cases this distinction is moot; e.g., the view that the name "keyboard" is associated with the keyboard input device is more or less equivalent to the view that "keyboard" identifies the handle connected (by the system) to that device. However, the naming of handles rather than their handlers has the effect of abstracting function, i.e., input-output behavior, from implementation; it supports our interposability desiderata, making (for example) interprocess communication indistinguishable from file I/O. Thus "date" names a handle which, when read, produces the current date; the name is associated with the communication path, and hence with the service performed, rather than (say) with an ASCII file containing the date. The "date" handle may in fact be effectively connected to a clock device or to a process which determines the date by interrogating a remote machine through the network.

Similarly, a directory is typically a domain which associates names with handles. Since it is accessed only via handles, however, the TRIX name semantics may be arbitrarily extended using an infinite variety of programs and communications which mimic the conventions of directories. For example, a compressed archive file A might be implemented to follow directory protocols, so that naive reference to "A/x" yields a handle which accesses (through a reformatting process) archive component "x" of A; archived files can then be transparently accessed by programs ignorant of the format -- or the existence -- of archives. In a distributed system, network gateways may map remote file systems into the local name space. Thus a reference to "/net/Joe/x" causes the local network server process, "/net", to forward a reference to "x" to a network server process on Joe's machine, which in turn accesses the handle "x" at the remote site. The effect is that the name graph on Joe's machine appears as a subgraph of the local graph rooted at /net/Joe.

Associated with each thread is a single stack, which is segmented by the kernel as the thread passes from one domain to another. This segmentation affords a level of protection, preventing the code in one domain from accessing data stacked in another; each thread conceptually has it has its own private stack for each domain.

This scheme is designed to exploit the efficiencies of conventional stack management (in particular, of the allocation of a single stack to each process) while effectively isolating domains from one another. Moreover, it conveniently provides for the stacking of privileged kernel data pertaining to a REQUEST on the region of a thread's stack corresponding to an inter-domain transition; this region, the kernel request frame, provides protected storage for previous state information as well as for the message itself. The protected storage of the message allows it to carry unforgeable system-protected capabilities, such as handles and data windows which extend the powers of the target domain.

5. VLSI TOOLS

Work by Terman continued on the logic-level timing simulator RSIM. Considerable time was spent comparing the predictions of RSIM with those of conventional circuit analysis techniques. Several changes in the underlying model were incorporated that greatly improved RSIM's accuracy; it is now possible to "tune" the simulation parameters so that RSIM's predictions are quite reasonable.

Previously RSIM modeled each transistor with a characteristic resistance determined by the size and implant of the transistor. This is still true, but in addition the context in which the transistor appears affects its characteristic resistance. For example, rather than having a single resistance for all enhancement devices, RSIM distinguishes pulldowns, source-followers, and pass devices as three distinct contexts and thus can model each separately with more accuracy.

The new model was tested at Digital Equipment Corporation which has started to use RSIM as the timing and logic-level simulation tool for several of its projects. In cooperation, we were able to adjust the simulation parameters so that RSIM's predictions agreed to within 10% of the circuit analysis results -- RSIM's advantages being greatly reduced computation requirements and the ability to simulate whole chips.

The interpretive front-end for RSIM, called RNL, and our textual schematic system NET were also improved, making them simpler to learn and use, and expanding their capabilities to include cMOS. A project to embed simulation and network primitives in a standard LISP (Franz Lisp under VAX/Unix) is now underway.

Finally, an investigation of special-purpose architectures for high-speed logic simulation was started. Several components of the project were completed:

1) A recognizer that develops a functional description of a circuit from a transistor network, i.e., there is an automatic translation from what the designer designed to a gate level description.

2) A prototype compiler which produces VAX code that directly computes the values for each network node. The compiler takes the functional description from step (1) as input, assigns an evaluation order to each operation, optimizes the sequence of operations (using constant propagation, etc.), and then produces code for the target machine (in this case a VAX).

3) The initial design of a highly-parallel simulation engine that might serve as an alternative target for the compilation process. This MIMD architecture offers the opportunity for performance improvements over conventional general-purpose SISD machines.

Work by Clark and Zippel has begun on the development of design aids for the development of high performance integrated circuits. Towards this end we are attempting build a system that makes available to an inexperienced designer as much of the design skills of a more experienced designer as possible. The simplest part of this system is a language for describing the topology of circuits in algorithmic, hierarchical manner. This language also incorporates a simple constraint system to allow for the specification of circuit parameters in a concise manner. It has been used to construct a flexible system that synthesizes conventional arithmetic logic units by Marc Rose.

A similar language has also been constructed for specifying signals in circuits. In this latter language, signals can be cascaded easily (they behave like programs) and again the parameters of the signals are controlled by a constraint system. This is especially powerful in the signal domain as it makes it easy to specify coincident and delayed edges, bootstrapped signals and concepts like noise and signal droop.

A 24x32 bit content-addressable memory chip design, reported previously, was fabricated and evaluated (Johnson). The initial chips worked correctly and robustly, tolerating power supply variations from 3.8 to 7.0 volts. They were somewhat slower than desired (210 ns), owing primarily to limitations imposed by the fabrication processes available to us.

Miscellaneous smaller VLSI-related projects include a simple register generator (Lakos) which sizes various component transistors in order to meet prespecified performance goals, and a 4kx1 static memory chip designed to operate under 100ns (Kramer).

Publications

1. Clark, D., et. al., "TRIX 1.0 Implementation Outline," Internal Memorandum, MIT Laboratory for Computer Science, Cambridge, MA, October 1981.

2. Clark, D., et. al., "The TRIX 1.0 Operating System," IEEE Distributed Processing Quarterly, 1,2 (December 1981).

3. Dertouzos, M.L., "Some Expected Computer Developments and Consequences: The Next Two Decades," to appear in Proc., New Telematics Service Day, E.A. Fiera Internazionale Milano Fair, Milan, Italy, April 1982.

4. Halstead, R.H., Jr., "Architecture of a Myriaprocessor," Advanced Computer Concepts, J.C. Solinsky, ed., La Jolla Institute, La Jolla, CA, 1981, pp. 107-132.

5. McLellan, H.R., and Ditzel, D.R., "Register Allocation for Free: The C Stack Cache," Sigplan Notices, 17,4 (April 1982), pp.48-56.

6. Zippel, Richard E., "Newton's Iteration and the Sparse Hensel Algorithm," Proc. SYMSAC '81 , Snowbird, Utah, August 1982.

Talks

1. Dertouzos, M.L., "Profitable Long-Range Industrial Research; A Response to the Japanese Fifth Generation Computer Challenge," MIT Corporate Executive Officers Meeting, hosted by Data General Corporation, Orlando, FL, February 1982.

2. Dertouzos, M.L. Moderator "Home Information Systems" MIT Alumni Association, Technology Day, MIT, Cambridge, MA, June 1982.

3. Dertouzos, M.L. "Introduction of the Laboratory for Computer Science and Distributed Systems," MIT/Siemens Conference on Distributed Systems , Siemens AG, Munich, W. Germany, May 1982.

4. Dertouzos, M.L. Panelist, IEEE Workshop on Communications Security, Crypto81, Santa Barbara, CA, August 1981.

5. Dertouzos, M.L. "Computers and People Beyond 1984," MIT Alumni Council, MIT Cambridge, MA, March 1982.

6. Dertouzos, M.L. "Computers: Developments and Consequences by the Year 2,000," Inaugural lecture upon election to membership, Athens Academy, Athens, Greece, February 1982.

7. Dertouzos, M.L. "The Japanese Fifth Generation Project: A Plan for World Supremacy in Informatics," Inaugural Board Meeting of Paris World Center, Paris, France, March 1982.

8. Dertouzos, M.L. "Expected Technological Developments in the Computer Field," MIT Industrial Liaison Program, European Course on Personal Computers, Networks, and Office Automation , Paris, France, February 1982.

9. Dertouzos, M.L. "The Information Marketplace," MIT Industrial Liaison Program, European Course on Personal Computers, Networks, and Office Automation , Paris, France, February 1982.

10. Dertouzos, M.L., "Ten Challenging Projects in Computer Science," University of Southern California, 10th Anniversary, Information Sciences Institute, Marina del Rey, CA, May 1982.

11. Dertouzos, M.L., Panelist, "Encryption: The Policy Issues," MIT Research Program on Communications Policy, MIT, Cambridge, MA, December 1981.

12. Dertouzos, M.L., "People and Computers by 2,000 A.D.," 2900 Club, Ltd. Conference on World Computing-- ICL's Role, London, England, November 1981.

13. Dertouzos, M.L., "Computer Science Research Directions," Kellogg National Fellowship Program, MIT Seminar Series, Cambridge, MA, November 1981.

14. Dertouzos, M.L., "Computer Applications in the Home," MIT Corporation Wives, MIT, Cambridge, MA, October 1981.

15. Halstead, R.H., Jr., "Object Oriented Multiprocessing," IBM/MIT Workshop, Lenox, MA, April 1982.

16. Halstead, R.H., Jr., "Object Oriented Multiprocessing," Lawrence Livermore Laboratory Workshop on Parallel Processing , New York, NY, April 1982.

17. Halstead, R.H., Jr., "Architecture of a Myriaprocessor," Invited Lecture, Texas Instruments Research Laboratories, Dallas, TX, July 1981.

18. Johnson, M.G., "A Content Addressable Memory for Virtual Address Translation," Bell Laboratories, MAC-32 design group, Holmdel, NJ, 9 April 1982.

19. Johnson, M.G., "An NMOS Content Addressable Memory," MIT VLSI Research Review, MIT, Cambridge, MA, 17 May 1982.

20. Mok, A., "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment," Invited Lecture, Harris Corporation, Melbourne, FL, 5 April 1982.

21. Mok, A., "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment," Intel Corporation, Portland, OR, 16 April 1982.

22. Terman, C.J., "Simulation Tools for LSI Design," Bell Laboratories, Murray Hill, NJ, November 1981.

23. Terman, C.J., "Introduction to NET and RSIM," 2 lectures, Digital Equipment Corporation, Hudson, MA, April 1982.

24. Terman, C.J., "Performance Estimation for Digital LSI Circuits," MIT VLSI Summer Course, Cambridge, MA, June 1982.

25. Terman, C.J., "A Tour Through the MIT Digital LSI Design Tools," Harris Corporation, Melbourne, FL, June 1982.

26. Ward, Stephen, "Introduction to Personal Computers," MIT Industrial Liaison Program, European Course on Personal Computers, Networks, and Office Automation , Paris, France, February 1982.

27. Zippel, Richard E., "Abstract Data Types, Flavors, and Capsules," Texas Instrument Corp., Dallas, TX, July 1981.

28. Zippel, Richard E., "Principles for High Performance MOS Design," Math Sciences Seminar, IBM Research Center, Yorktown Heights, NY, December 1981.

29. Zippel, Richard E., "SCHEMA: A System for High Performance MOS Design," University of California, Berkeley, February 1982.

30. Zippel, Richard E., "Static RAM's and Design Systems," Digital Equipment Corporation, Hudson, MA, March 1982.

31. Zippel, Richard E., "A VLSI Workstation," Siemens Corporation AG, Munich, W. Germany, May 1982.

32. Zippel, Richard E., "An Undergraduate Course in Digital MOS Circuits," MIT VLSI Research Review , May 1982.

Theses Completed

1. Alcabes, Harvey, "Syntactic Error Recovery in an LALR Parser," S.M. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., completed July 1981.

2. Esposito, Dan, "An Algorithm for Efficient Digital Logic Simulation," S.M. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., completed August 1981.

3. Goldman, Seth, "A Digital Integrated Circuit Design for Conway's Life Game and General Cellular Automata," S.B. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., completed May 1982.

4. Hankins, Brenda, "Path Expressions-- An Extension to MuLisp," S.B. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., completed May 1982.

5. Holmes, Kirk, "Implementation of Speech Recognition in a Microprocessor Environment," S.B. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., completed May 1982.

6. Hughes III, T. W. L., "Othello," S.B. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., completed May 1982.

7. Kesselman, Joseph, "PROTEUS -- A Microprogrammable, Multiprocessor Computer," S.B. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., completed August 1981.

8. Lakos, John, "The Development of an nMOS VLSI Register Array

Generator: RAG," S.B. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., completed May 1982.

9. Le, Cahn, "A Deterministic Schedule for Process Control Computations," S.B. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., completed May 1982.

10. Lukac, Tibor, "A Color Graphics Terminal Incorporating a New Generation Computer," S.B. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., completed May 1982.

11. McMahon, Douglas, "Real-Time Acquisition of Signal Data to VAX 11/780 Database," S.B. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., completed May 1982.

12. Nahabedian, Markar, "Considerations for the Design of a Command Processor for the TRIX Operating System," S.B. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., completed May 1982.

13. Ronkin, Nelson, "The Use of Futures in Parallel Processing Language Constructs," S.B. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., completed May 1982.

14. Rose, Marc, "A High Level Tool for the Computer-Aided-Design of ALU Circuitry," S.B. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., completed May 1982.

15. Vogel, Dan, "Real-Time Mapping of Signal Processing Jobs onto Multiprocessor Networks," S.B./S.M. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., completed May 1982.

Theses in Progress

1. Anderson, T., "The Design of a Multiprocessor Development System," S.M. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., expected August 1982.

2. Goddeau, D. "Learning and Adaptation in Large Scale Processor Arrays," S.M. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., expected December 1982.

3. Johnson, M., "Efficient Modeling for Short-Channel MOS Circuit Simulation," S.M. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., expected August 1982.

4. Lake, A., "Evaluation of Multiprocessor Communications Architectures," S.M. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., expected September 1982.

5. Mandry, J., "A Novel Priority State Controller for a Computer Bus," S.M. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., expected December 1982.

6. Mok, A., "Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environments," PhD. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., expected September 1982.

7. Powell, J. "Voice Entry for Interactive Control of a Small Computer," S. M. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., expected December 1982.

8. Sieber, J. "TRIX: An Operating System Supporting Network Communications," S.M. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., expected January 1982.

9. Teixeira, T.J. "Compiling Programs to Meet Performance Requirements," PhD. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., expected December 1982.

10. Terman, C. J. "Simulation Tools for LSI Design," PhD. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., expected August 1982.

11. Troisi, J., "An Interpreter and Symbolic Debugger for C," S.M. thesis, MIT Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., expected December 1981.

SYSTEMATIC PROGRAM DEVELOPMENT

Academic Staff

J. Guttag, Group Leader

Graduate Students

S. Atreya	R. Forgaard
R. Kownacki	M. Srivas
J. Wing	J. Zachary

Undergraduate Students

D. Detlefs

Support Staff

E. Pothier

Postdoctoral Fellow

P. Lescanne

1. INTRODUCTION

Our objective is to facilitate the useful application of precise specifications to the development and maintenance of software. We believe the absence of adequate tools for constructing and using specifications is the most significant bottleneck in this area.

Until an implementation exists, a programming language can never be of more than academic interest. The same is true for specification languages. To design a programming language without due regard for implementation issues is a serious mistake. The same is true for specification languages. Despite these truisms, relatively little attention has paid to the issue of implementing specification languages. We believe that this, more than anything else, is what has kept formal specification languages from becoming a practical tool in the development of software. A distinguishing aspect of our work is the emphasis on the development of an integrated set of mental and software tools.

Part of the problem has been a failure to come to grips with just what it should mean to "implement" a specification language. Our efforts in this area have been strongly influenced by three critical working hypotheses:

1) Almost all of the advantages of formal specifications over informal ones rest upon the amenability of formal specifications to machine analysis and manipulation,

2) Many of the benefits of any kind of specification come during the process of constructing the specification, and

3) Most of the time one must deal with partial, rather than completed, specifications.

2. SPECIFICATION LANGUAGES

2.1. Interface Languages

The semantics of an interface language is dependent on the semantics of the target implementation language. Currently, CLU is being used as a sample target implementation language. A suitable language for writing interfaces for CLU has been developed. The three kinds of interface specifications in her model are procedural, data, and iterator interfaces, corresponding to procedures, clusters, and iterators in CLU. Procedural interfaces use types provided by data interfaces. These types are defined by sorts defined in traits.

An example of a procedural interface is as follows:

```
interface SetProc

imports SetDa          % SetDa provides the type Set.
uses setTrait
provides

        % This procedure possibly mutates its second argument.
        Intersect = proc (s1: Set, s2: Set)
            pre true
            modifies s2
            post for all i [has(s2'↑,i) == has(s1↑,i) and
                                has(s2↑,i)]
        end Intersect

end SetProc
```

An identifier, x, denotes an object; in the pre- and post-conditions, x denotes the initial state, x' denotes the final state); x↑ denotes the value of the object x. All values of objects are expressible as a ground term in the trait language. A procedural interface heading looks similar to that of procedures in CLU. Pre- and post-conditions contain assertions on the state of the CLU universe before and after the call to the procedure. For the above example, the value of the set object s2 changes to become the intersection of the initial values of the set objects s1 and s2.

An example of a data interface, providing the type Set is as follows:

```
interface SetDa

imports IntegerDa
uses setTrait
provides
 Set = type (set, Singleton, Union, Member)

 Singleton = proc (i: Integer) returns (s: Set)
    pre true
    post s'↑ == insert(empty, i↑)
 end Singleton

 % This procedure doesn't mutate either of its input arguments.
 Union = proc (s1: Set, s2: Set) returns (s3: Set)
     pre true
     post for all i [has(s3'↑,i) == has(s1↑,i) or has(s2↑,i)]
 end Union

 Member = proc (s: Set, i: Integer) returns (b: Bool)
     pre true
     post b'↑ == has(s↑, i↑)
 end Member

end SetDa
```

The meaning of the first line after "provides" is that a type induction rule for the type Set is defined by the sort set (obtained from the trait setTrait). The induction rule is

$$\frac{\mathtt{P(Singleton(i)),\ P(s1)\ and\ P(s2)\ implies\ P(Union(s1,s2))}}{\mathtt{P(s)}}$$

where i is of type Integer, s1, s2, and s are of type Set, and P is an arbitrary predicate in first order predicate logic with equality.

An example of an iterator interface is as follows:

```
interface ElemsIter

imports integerDa

          Elems = iter (s: Set) yields (i: Integer)
            let n = nth iteration in
              pre true
              modifies s
              post has(sn↑, i↑) and sn'↑ == delete(sn↑,i↑)

              pre isEmpty(sn↑)
              post return
          end Elems

    end ElemsIter
```

Since the arguments of an iterator can be mutated by the iterator or the loop body invoking the iterator, we must be able to denote intermediate states of objects. We use "n" above as this "loop variable" so that "sn" denotes the object s at the beginning of the nth invocation of Elems and "sn'" denotes the object s at the end.

3. THE SPECIFICATION EDITOR

We designed our shared specification language in tandem with a sophisticated editing and perusing tool. Many language design decisions were influenced by the presumption that specifications would be produced and read interactively using an editor that would provide "real time" feedback. The first design is almost complete [1], and we plan to begin implementation in the fall. The primary functions performed by the editor are:

1) Supplying templates -- These are the basic building blocks of specifications,

2) Generation of redundant information -- Information that will make the specification easier to read,

3) Generation of different views -- It is often impossible for the author of a specification to predict the level of detail that various readers will want to see. The editor allows readers to control this,

4) Checking -- Our goal is to catch mistakes early. Experience indicates that superficial errors, e.g., type errors, are often indicative of serious underlying confusion, and catching them early is a valuable service,

5) Keeping track of missing information -- Supplying the user with a list of things that still need to be done, e.g., inconsistencies to be resolved or what information is necessary to complete a specification,

6) Specification to specification mappings -- A collection of syntactic mechanisms for doing semantically significant things, e.g., abstracting, renaming an operation, adding properties, or modifying a signature.

The editor provides comprehensive support for all phases of the construction and use of specifications. Because of the high level of machine interaction that we feel is essential to the specification process, we intend that the editor will be employed as the sole means of performing specification tasks. We further expect that our shared specification language will evolve along with the editor over the course of time.

We intend to establish a strong coupling between the language and its editor. Because of this coupling, linguistic features that might otherwise be impractical are rendered feasible. The design of Shroff has already been influenced to some extent by the expectation of specialized editor support; future language changes can be expected to be influenced to an even greater extent. Evidence of this influence can be found in the techniques of specification construction encouraged by the textually oriented semantics of Bicycle.

A Bicycle specification is a large body of trait, sort, variable, and function names composed together. The acceptable compositions are defined by a variety of syntactic rules. The intentional overloading and controlled collision of function and sort names is an important technique of composition. Because of this, the exact meaning of a name can in general be determined only through careful analysis of its context. The amount of analysis required to validate compositions and examine contexts has proven to be overly taxing when performed by hand; indeed, it has limited the size and complexity of specifications we have been able to construct. Were it not for the prospect of machine support, the language would require extensive redesign.

The basic Bicycle constructs--"include," "assume," "without," and "bind"--are defined as textual transformations upon the traits in which they appear. It is sometimes desirable to actually to perform a transformation in order to understand its ramifications; conversely, it is sometimes useful to structure the complexity of a trait by introducing a simplifying construct. Editor support in transacting the details of these transformations is essential.

In the case of consistency checking, the editor serves as a sort checker for the Bicycle language; in the case of the transformation constructs, it acts as a language interpreter. In both cases, features of the language are closely related to features of the editor. The distinction between language and editor should gradually disappear.

The set of abstract operations that the editor provides upon specifications is an important part of its design. However, we believe that the details of the user interface--the means by which these abstract operations are supplied--are an equally important consideration. A carefully designed user interface is essential to attaining the goal of restricting all manipulations of specifications to the editor, since specifiers will be more likely to embrace a tool that is convenient to use. We also believe that the style of the editor's user interface will strongly affect the style of specifications written using it.

Although the two activities can be considered separately, the editor draws no hard distinction between the phases of reading and writing specifications. The task of specification generally involves both forms of interaction; the difference between the two is more a matter of point of view than point of fact. For example, a specifier might be reading a specification in order to modify it; similarly, a creator would certainly require the tools available to readers in order to evaluate his work. Separate means of reading and writing would only serve to reinforce an illusory distinction; hence, a uniform user interface is presented at all times.

The user interface is designed for implementation upon a high-resolution display terminal equipped with a pointing device. A high-resolution display permits a more expressive interface because of the flexibility it possesses, while a pointing device enables a convenient, rapid mode of pointing access to all text on the display. No pains have been taken to enable a graceful realization of our design upon more conventional display terminals. Although high-resolution terminals are presently common only in research environments, we believe that they represent, as do formal specifications, the technology of choice in the near future. The advantages of such terminals outweigh the temporary drawback of availability. We wish to exploit these advantages to the fullest.

We choose to ground the display upon a simple windowing mechanism featuring a fixed number of windows. We make this decision for two reasons. First ,we wish to free the specifier from the overhead of managing large systems of windows. We are then able to fix the content type and interpretation of each window. By controlling the methods of displaying specifications, we will be better able to encourage a particular editing methodology.

The editor provides its comprehensive support on an incremental basis. The specifier does not independently create text and periodically request the editor to process it, as is the case in most programming environments. Instead, the editor provides full support throughout all phases of editing. To realize this goal, the editor is syntax-directed. It prohibits the introduction of syntactic errors where possible, and incrementally detects and flags them where not.

The Bicycle specification language does not provide for syntactically correct intermediate forms; neither do specifications spring into existence in a completed form. In order to view developing specifications syntactically, we must extend the Bicycle syntax. We introduce a template oriented syntax for Bicycle, in which generic stencils may stand in the place of incomplete portions of that text. The process of creating a specification then becomes one of successively refining templates.

One benefit of syntax-directed editing is the close involvement by the editor which is possible during specification creation. Most syntax errors, by definition, are impossible to commit; the remainder are immediately highlighted. The editor can present a correctly formatted version of the text at all times. Work that remains to be done can be noted and displayed to the specifier. Also, the editor can automatically supply keywords and symbols as parts of templates. These possibilities serve to reinforce the close relationship between the language and the editor, blurring the distinction between the two even further.

The derivation process of specifications is rooted in an underlying system of templates. We hold, however, that a style of text entry in which the user directly and transparently performs expansion operations upon templates is unacceptable. Instead, we provide a style of entry that allows the user to enter specification text much as he would were he typing it as free text. The editor examines the text which is entered, and performs template modifications as required. This style of editing is more natural, and therefore more likely to be embraced by specifiers.

The editor's syntax-directed interface contributes to the ease of initially creating specifications. Our editor will provide more, however, than a mere means of creating specifications. Support will be provided for reading, analyzing, and otherwise manipulating specification text. A rich set of high-level operations will be provided for this purpose. The effects of the invocation of a high-level operation are likely to be extensive. It is essential that specifiers be able to approach these operations with a confidence that their invocation will not cause irrevocable damage. Thus, the effects of all high-level operations are designed to be reversible.

Because of the textual motivation of the Bicycle constructs, operations for interpreting the constructs are an important aid to reading specifications. Operations upon the equations that ultimately define the abstract meaning of specification are also necessary. Such operations will, of course, involve the use of the theorem proving subsystem. Operations which analyze the structure of specifications, perhaps providing objective judgments of abstract qualities, will provide a means of supporting particular specification styles. As we gain experience with the craft of specification using an editor, we expect that other such high-level operations will be clarified. A means for their invocation must accordingly be designed into the editor.

The arguments to high-level operations are syntactically meaningful portions of specification text. The editor provides a means of selecting those portions which are intended to be arguments. Because the set of high-level operations is likely to be both large and changing over time, a menu oriented style of invocation is provided.

4. THE SPECIFICATION LIBRARY

In order to deal effectively with the difficulties of constructing system and design specifications, a specifier requires a well organized library of primitive abstractions. We believe that the effort of specifying large software projects can be significantly reduced by the existence of "building block" specifications which may be borrowed, modified or instantiated as the specifier requires. The key issues involved in a library system are convenience of use, provisions for growth and maintenance, and compatibility with our overall specification environment, particularly the editor.

Throughout the past year, Sriram Atreya focused his research effort on the initial design for such a library [2]. He employed formal specifications as a tool in the presentation of his design. We profited from his experience in writing a large scale specification as well as by the actual product of his effort.

Experience with various programming libraries and other filing systems suggests that a primary design consideration should be ease of use. The knowledge that a needed abstraction exists is little consolation to a specifier if the method for obtaining it is unclear. In order to deal with this problem, Atreya employs a method which is a generalization of the common card catalog system utilized for libraries of books. This involves the creation of an appropriate *classification scheme* combined with a set of operators which permit a natural *browsing process* based on the scheme.

The second major design goal arises from the consideration that a specification library will certainly undergo frequent changes and will probably exhibit a steady growth. A library must be designed with the concept of a long lifetime in mind. Atreya addresses these problems by rejecting a static classification scheme in order to maintain the balance of the library and to enhance its flexibility.

5. THE REWRITE RULE LABORATORY

An important benefit of formal specifications lies in the possibilities of checking for consistency and constructing formal proofs of interesting properties. Naturally, we would prefer to automate as much of this process as possible. Term rewriting systems may be employed to obtain decision procedures for equational theories, the formal basis of our approach to specifications. The principle goal of our rewrite rule laboratory is to offer the power of term rewriting systems to the specifier without requiring any particular knowledge of this model of computation. Pierre Lescanne has successfully implemented the kernel of the laboratory, which we will discuss below.

Two properties which are required by rewriting systems for theorem proving are uniform termination and confluence. Uniform termination means that all rewriting paths are finite. Confluence implies that whenever different rules are applied to the same term, it is possible for the resultant rewriting paths to converge to one term. When a rewriting system exhibits both of these properties, it is said to be convergent. In order to use term rewriting systems to prove equalities in equational theories, we require the systems to be convergent.

An extension of the Knuth-Bendix algorithm allows us to mechanically associate a convergent system of rewrite rules with a set of equations. This algorithm produces equations which a specific ordering algorithm orients to make rewrite rules. Recursive decomposition ordering, given in [3], is an algorithm for proving uniform termination of a rewriting system. The algorithm is based on a careful analysis of the positions of operation symbols in terms to determine a precedence relation among the operators. The behavior of the algorithm has been studied in various simple cases, [4][5], as well as in the case of monadic terms on a totally ordered set of symbols [6].

In February, Lescanne began work on an implementation of REVE, a software system which manipulates rewrite rules and implements the Knuth-Bendix procedure. REVE uses Knuth-Bendix to automatically find a convergent set of rewrite rules corresponding to a set of equations when it is possible to do so. Once a convergent term writing system has been obtained, REVE can use Knuth-Bendix to prove theorems that normally require the use of induction.

He has tested it by proving some fairly difficult results in the axiomatization of groups. Among these results were questions posed by Knuth and Bendix in their paper. As an example, one of the problems solved by REVE is the following from [7]: "Prove that binary algebras which satisfy the unique equation

$$x / ((((x / x) / y) / z) / (((x / x) / x) / z) = y$$

where / is right division, are groups."

The interesting feature of REVE lies in the integration of term orderings with a Knuth-Bendix algorithm. These orderings are Decomposition Ordering [3] and Recursive Path Ordering [8][9] This combination explains REVE's success in proving results in the axiomatization of groups which were never before proven by machine. The Knuth-Bendix algorithm used in REVE is from Huet [10] and [11]. the unification algorithm is from Martelli and Montanari

6. INTERACTIONS OUTSIDE OF LCS

The systematic program development group has make a point of maintaining close contact with researchers outside of MIT. In particular, J. Guttag collaborates with Jim Horning of Xerox PARC, Dave Musser and Deepak Kapur of General Electric Corporate Research and Development, and John Williams of IBM Research at San Jose. J. Wing also has done work at Xerox PARC [12][13].

References

1. Zachary, J.L. "A Syntax-Directed Specification Editor," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

2. Atreya, S. "Formal Specification of a Specification Library," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

3. Jouannaud, J-P., Lescanne, P. and Reinig, F. "Recursive Decomposition Ordering," Conf. on Formal Description on Programming Concepts, Garmisch-Partenkirchen, Germany, 1982.
Reinig"

4. Lescanne, P. "Decomposition Ordering as a Tool To Prove the Termination of Rewriting Systems," Seventh IJCAI, Vancouver, Canada, 1981, 548-550.

5. Lescanne, P. "Some Properties of Decomposition Ordering," Symposium AFCET "The Mathematics for Computer Science, Paris, March 1982.

6. Lescanne, P. "Two Implementations of the Recursive Path Ordering On Monadic Terms," 19th Annual Allerton Conf. on Communication, Control and Computing, Allerton House, Monticello, Illinois, 1981, 634-643.

7. Higman, G. and Neumann, B. H. "Groups as Groupoids With One Law," Publ. Math. Debrecen., (2), 1952, 215-221.

8. Dershowitz, N. and Manna, Z. "Proving Termination With Multiset Orderings," Journal of the CACM, 22, 1979, 465-476.

9. Kamin, S. and Levy, J.J. "Attempts For Generalizing the Recursive Path Ordering," to appear.

10. Huet, G. "A Complete Proof of Correctness Of The Knuth-Bendix Completion Algorithm," J. Comp. Sys. Sc., (23), 1981, 11-21.

11. Martelli, A. and Montanari, U. An Efficient Unification Algorithm," ACM Trans. Program. Lang. Syst., (4) 1982, 258-282.

12. Wing, J.M. "Thoughts on Writing Specifications," Xerox Internal Memo, Xerox Palo Alto Research Center, Palo Alto, CA, August 1981.

13. Wing, J.M. "Building Specifications Using PIE," Xerox Internal Memo, Xerox Palo Alto Research Center, Palo Alto, CA, August 1981.

Publications

1. Guttag, J. V., Kapur, D. and Musser, D. R. "On Proving Uniform Termination and Restricted Termination of Rewriting Systems," to appear in SIAM Journal of Computing

2. Guttag, J.V., Horning, J.J., and Wing, J.M. "On Putting Formal Specifications to Productive Use," to appear in Science of Programming.

3. Guttag, J.V., Horning, J.J., and Williams, J. "FP with Data Abstraction and Strong Typing," Proceedings of a Conference on Functional Programming and Computer Architecture, October 1981.

4. Guttag, J. V., Kapur, D. and Musser, D. R. "Derived Pairs, Overlap Closures, and Rewrite Dominoes: New Tools for Analyzing Term Rewriting Systems," to appear in Proceedings of the 1982 ICALP Conference, July 1982.

5. Guttag, J.V. "Notes on Using Types and Type Abstraction in Functional Programming," Functional Programming and its Applications, North Holland (1982).

6. Guttag, J.V. "A Few Remarks on Putting Formal Specifications to Productive Use," Proceedings of a Workshop on Program Specification, Lecture Notes in Computer Science 134, Springer-Verlag (1982).

7. Jouannaud, J.P. and Lescanne, P. "On Multiset Orderings," to appear in Inform. Proc. Ltrs.

8. Jouannaud, J-P., Lescanne, P. and Reinig, F. "Recursive Decomposition Ordering," Conf. on Formal Description on Programming Concepts, Garmisch-Partenkirchen, Germany, 1982.

9. Lescanne, P. "Decomposition Ordering as a Tool To Prove the Termination of Rewriting Systems," Seventh IJCAI, Vancouver, Canada, 1981, 548-550.

10. Lescanne, P. "Two Implementations of the Recursive Path Ordering On Monadic Terms," 19th Annual Allerton Conf. on Communication, Control and Computing, Allerton House, Monticello, Illinois, 1981, 634-643.

11. Lescanne, P. "Some Properties of Decomposition Ordering," Symposium AFCET "The Mathematics for Computer Science, Paris, March 1982.

Theses Completed

1. Atreya, S. "Formal Specification of a Specification Library," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

2. Zachary, J.L. "A Syntax-Directed Specification Editor," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1982.

Theses in Progress

1. Wing, J.M. "Bridging Algebraic Specifications and Their Implementations Via Interfaces," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1982.

Talks

1. Lescanne, P., "Decomposition Ordering as a Tool to Prove the Termination of Rewriting Systems," 7th International Joint Conference on Artificial Intelligence, August 27, 1981.

2. Lescanne, P., "Two Implementations of Recursive Path Ordering on Monadic Terms," 19th Annual Allerton Conf. on Communication, Control, and Computing, Allerton House, Monticello, Illinois, October 1, 1981.

3. Lescanne, P., "Decomposition Ordering, a Kind of Simplification Ordering Used to Prove Termination of Term Rewriting Systems," General Electric Research and Development Center, May 10, 1982.

4. Lescanne, P., "Decomposition Ordering, a Kind of Simplification Ordering Used to Prove Termination of Term Rewriting Systems," MIT Laboratory for Computer Science, June 1, 1982.

5. Lescanne, P., "Decomposition Ordering, a Kind of Simplification Ordering Used to Prove Termination of Term Rewriting Systems," Bell Laboratories, Murray Hill, NJ, June 10, 1982.

PUBLICATIONS

Technical Memoranda

TM-10[6]	Jackson, James N. Interactive Design Coordination for the Building Industry, June 1970, AD 708-400
TM-11	Ward, Philip W. Description and Flow Chart of the PDP-7/9 Communications Package, July 1970, AD 711-379
TM-12	Graham, Robert M. File Management and Related Topics June 12, 1970, September 1970, AD 712-068
TM-13	Graham, Robert M. Use of High Level Languages for Systems Programming, September 1970, AD 711-965
TM-14	Vogt, Carla M. Suspension of Processes in a Multi-processing Computer System, September 1970, AD 713-989
TM-15	Zilles, Stephen N. An Expansion of the Data Structuring Capabilities of PAL, October 1970, AD 720-761
TM-16	Bruere-Dawson, Gerard Pseudo-Random Sequences, October 1970, AD 713-852
TM-17	Goodman, Leonard I. Complexity Measures for Programming Languages, September 1971, AD 729-011
TM-18	Reprinted as TR-85
TM-19	Fenichel, Robert R. A New List-Tracing Algorithm, October 1970, AD 714-522

[6] TMs 1-9 were never issued.

TM-20	Jones, Thomas L. A Computer Model of Simple Forms of Learning, January 1971, AD 720-337
TM-21	Goldstein, Robert C. The Substantive Use of Computers For Intellectual Activities, April 1971, AD 721-618
TM-22	Wells, Douglas M. Transmission Of Information Between A Man-Machine Decision System And Its Environment, April 1971, AD 722-837
TM-23	Strnad, Alois J. The Relational Approach to the Management of Data Bases, April 1971, AD 721-619
TM-24	Goldstein, Robert C. and Alois J. Strnad The MacAIMS Data Management System, April 1971, AD 721-620
TM-25	Goldstein, Robert C. Helping People Think, April 1971, AD 721-998
TM-26	Iazeolla, Giuseppe G. Modeling and Decomposition of Information Systems for Performance Evaluation, June 1971, AD 733-965
TM-27	Bagchi, Amitava Economy of Descriptions and Minimal Indices, January 1972, AD 736-960
TM-28	Wong, Richard Construction Heuristics for Geometry and a Vector Algebra Representation of Geometry, June 1972, AD 743-487
TM-29	Hossley, Robert and Charles Rackoff The Emptiness Problem for Automata on Infinite Trees, Spring 1972, AD 747-250
TM-30	McCray, William A. SIM360: A S/360 Simulator, October 1972, AD 749-365
TM-31	Bonneau, Richard J. A Class of Finite Computation Structures Supporting the Fast Fourier Transform, March 1973, AD 757-787

TM-32	Moll, Robert An Operator Embedding Theorem for ComplexityClasses of Recursive Functions, May 1973, AD 759-999
TM-33	Ferrante, Jeanne and Charles Rackoff A Decision Procedure for the First Order Theory of Real Addition with Order, May 1973, AD 760-000
TM-34	Bonneau, Richard J. Polynomial Exponentiation: The Fast Fourier Transform Revisited, June 1973, PB 221-742
TM-35	Bonneau, Richard J. An Interactive Implementation of the Todd-Coxeter Algorithm, December 1973, AD 770-565
TM-36	Geiger, Steven P. A User's Guide to the Macro Control Language, December 1973, AD 771-435
TM-37	Schonhage, A. Real-Time Simulation of Multidimensional Turing Machines by Storage Modification Machines, December 1973, PB 226-103/AS
TM-38	Meyer, Albert R. Weak Monadic Second Order Theory of Succesor is not Elementary-Recursive, December 1973, PB 226-514/AS
TM-39	Meyer, Albert R. Discrete Computation: Theory and Open Problems, January 1974, PB 226-836/AS
TM-40	Paterson, Michael S., Michael J. Fischer and Albert R. Meyer An Improved Overlap Argument for On-Line Multiplication, January 1974, AD 773-137
TM-41	Fischer, Michael J. and Michael S. Paterson String-Matching and Other Products, January 1974, AD 773-138
TM-42	Rackoff, Charles On the Complexity of the Theories of Weak Direct Products, January 1974, PB 228-459/AS

TM-43	Fischer, Michael J. and Michael O. Rabin Super-Exponential Complexity of Presburger Arithmetic, February 1974, AD 775-004
TM-44	Pless, Vera Symmetry Codes and their Invariant Subcodes, May 1974, AD 780-243
TM-45	Fischer, Michael J. and Larry J. Stockmeyer Fast On-Line Integer Multiplication, May 1974, AD 779-889
TM-46	Kedem, Zvi M. Combining Dimensionality and Rate of Growth Arguments for Establishing Lower Bounds on the Number of Multiplications, June 1974, PB 232-969/AS
TM-47	Pless, Vera Mathematical Foundations of Flip-Flops, June 1974, AD 780-901
TM-48	Kedem, Zvi M. The Reduction Method for Establishing Lower Bounds on the Number of Additions, June 1974, PB 233-538/AS
TM-49	Pless, Vera Complete Classification of (24,12) and (22,11) Self-Dual Codes, June 1974, AD 781-335
TM-50	Benedict, G. Gordon An Enciphering Module for Multics, S.B. Thesis, EE Dept., July 1974, AD 782-658
TM-51	Aiello, Jack M. An Investigation of Current Language Support for the Data Requirements of Structured Programming, S.M. & E.E. Thesis, EE Dept., September 1974, PB 236-815/AS
TM-52	Lind, John C. Computing in Logarithmic Space, September 1974, PB 236-167/AS
TM-53	Bengelloun, Safwan A. MDC-Programmer: A Muddle-to Datalanguage Translator for Information Retrieval, S.B. Thesis, EE Dept., October 1974, AD 786-754

TM-54 Meyer, Albert R.
The Inherent Computation Complexity of Theories of Ordered Sets: A Brief Survey, October 1974, PB 237-200/AS

TM-55 Hsieh, Wen N., Larry H. Harper and John E. Savage
A Class of Boolean Functions with Linear Combinatorial Complexity, October 1974, PB 237-206/AS

TM-56 Gorry, G. Anthony
Research on Expert Systems, December 1974

TM-57 Levin, Michael
On Bateson's Logical Levels of Learning, February 1975

TM-58 Qualitz, Joseph E.
Decidability of Equivalence for a Class of Data Flow Schemas, March 1975, PB 237-033/AS

TM-59 Hack, Michel
Decision Problems for Petri Nets and Vector Addition Systems, March 1975 PB 231-916/AS

TM-60 Weiss, Randell B.
CAMAC: Group Manipulation System, March 1975, PB 240-495/AS

TM-61 Dennis, Jack B.
First Version of a Data Flow Procedure Language, May 1975

TM-62 Patil, Suhas S.
An Asynchronous Logic Array, May 1975

TM-63 Pless, Vera
Encryption Schemes for Computer Confidentiality, May 1975, AD A010-217

TM-64 Weiss, Randell B.
Finding Isomorph Classes for Combinatorial Structures, S.M. Thesis, EE Dept., June 1975

TM-65 Fischer, Michael J.
The Complexity Negation-Limited Networks - A Brief Survey, June 1975

TM-66	Leung, Clement Formal Properties of Well-Formed Data Flow Schemas. S.B., S.M. & E.E. Thesis, EE Dept., June 1975
TM-67	Cardoza, Edward E. Computational Complexity of the Word Problem for Commutative Semigroups, S.M. Thesis, EE & CS Dept., October 1975
TM-68	Weng, Kung-Song Stream-Oriented Computation in Recursive Data Flow Schemas, S.M. Thesis, EE & CS Dept., October 1975
TM-69	Bayer, Paul J. Improved Bounds on the Costs of Optimal and Balanced Binary Search Trees, S.M. Thesis, EE & CS Dept., November 1975
TM-70	Ruth, Gregory R. Automatic Design of Data Processing Systems, February 1976, AD A023-451
TM-71	Rivest, Ronald On the Worst-Case of Behavior of String-Searching Algorithms, April 1976
TM-72	Ruth, Gregory R. Protosystem I: An Automatic Programming System Prototype, July 1976, AD A026-912
TM-73	Rivest, Ronald Optimal Arrangement of Keys in a Hash Table, July 1976
TM-74	Malvania, Nikhil The Design of a Modular Laboratory for Control Robotics, S.M. Thesis, EE & CS Dept., September 1976, AD A030-418
TM-75	Yao, Andrew C. and Ronald I. Rivest K + 1 Heads are Better than K, September 1976, AD A030-008
TM-76	Bloniarz, Peter A., Michael J. Fischer and Albert R. Meyer A Note on the Average Time to Compute Transitive Closures, September 1976

TM-77	Mok, Aloysius K. Task Scheduling in the Control Robotics Environment, S.M. Thesis, EE & CS Dept., September 1976, AD A030-402
TM-78	Benjamin, Arthur J. Improving Information Storage Reliability Using a Data Network, S.M. Thesis, EE & CS Dept., October 1976, AD A033-394
TM-79	Brown, Gretchen P. A System to Process Dialogue: A Progress Report, October 1976, AD A033-276
TM-80	Even, Shimon The Max Flow Algorithm of Dinic and Karzanov: An Exposition, December 1976
TM-81	Gifford, David K. Hardware Estimation of a Process' Primary Memory Requirements, S.B. Thesis, EE & CS Dept., January 1977
TM-82	Rivest, Ronald L., Adi Shamir and Len Adleman A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, April 1977, AD A039-036
TM-83	Baratz, Alan E. Construction and Analysis of Network Flow Problem which Forces Karzanov Algorithm to $O(n^3)$ Running Time, April 1977
TM-84	Rivest, Ronald L. and Vaughan R. Pratt The Mutual Exclusion Problem for Unreliable Processes, April 1977
TM-85	Shamir, Adi Finding Minimum Cutsets in Reducible Graphs, June 1977, AD A040-698
TM-86	Szolovits, Peter, Lowell B. Hawkinson and William A. Martin An Overview of OWL, A Language for Knowledge Representation, June l977, AD A041-372
TM-87	Clark, David., editor Ancillary Reports: Kernel Design Project, June 1977

TM-88	Lloyd, Errol L. On Triangulations of a Set of Points in the Plane, S.M. Thesis, EE & CS Dept., July 1977
TM-89	Rodriguez, Humberto Jr. Measuring User Characteristics on the Multics System, S.B. Thesis, EE & CS Dept., August 1977
TM-90	d'Oliveira, Cecilia R. An Analysis of Computer Decentralization, S.B. Thesis, EE & CS Dept., October 1977, AD A045-526
TM-91	Shamir, Adi Factoring Numbers in O(log n) Arithmetic Steps, November 1977, AD A047-709
TM-92	Misunas, David P. Report on the Workshop on Data Flow Computer and Program Organization, November 1977
TM-93	Amikura, Katsuhiko A Logic Design for the Cell Block of a Data-Flow Processor, S.M. Thesis, EE & CS Dept., December 1977
TM-94	Berez, Joel M. A Dynamic Debugging System for MDL, S.B. Thesis, EE & CS Dept., January 1978, AD A050-191
TM-95	Harel, David Characterizing Second Order Logic with First Order Quantifiers, February 1978
TM-96	Harel, David, Amir Pnueli and Jonathan Stavi A Complete Axiomatic System for Proving Deductions about Recursive Programs, February 1978
TM-97	Harel, David, Albert R. Meyer and Vaughan R. Pratt Computability and Completeness in Logics of Programs, February 1978

TM-98	Harel, David and Vaughan R. Pratt Nondeterminism in Logics of Programs, February 1978
TM-99	LaPaugh, Andrea S. The Subgraph Homeomorphism Problem, S.M. Thesis, EE & CS Dept., February 1978
TM-100	Misunas, David P. A Computer Architecture for Data-Flow Computation, S.M. Thesis, EE & CS Dept., March 1978, AD A052-538
TM-101	Martin, William A. Descriptions and the Specialization of Concepts, March 1978, AD A052-773
TM-102	Abelson, Harold Lower Bounds on Information Transfer in Distributed Computations, April 1978
TM-103	Harel, David Arithmetical Completeness in Logics of Programs, April 1978
TM-104	Jaffe, Jeffrey The Use of Queues in the Parallel Data Flow Evaluation of "If-Then-While" Programs, May 1978
TM-105	Masek, William J. and Michael S. Paterson A Faster Algorithm Computing String Edit Distances, May 1978
TM-106	Parikh, Rohit A Completeness Result for a Propositional Dynamic Logic, July 1978
TM-107	Shamir, Adi A Fast Signature Scheme, July 1978, AD A057-152
TM-108	Baratz, Alan E. An Analysis of the Solovay and Strassen Test for Primality, July 1978
TM-109	Parikh, Rohit Effectiveness, July 1978

PUBLICATIONS

TM-110	Jaffe, Jeffrey M. An Analysis of Preemptive Multiprocessor Job Scheduling, September l978
TM-111	Jaffe, Jeffrey M. Bounds on the Scheduling of Typed Task Systems, September l978
TM-112	Parikh, Rohit A Decidability Result for a Second Order Process Logic, September 1978
TM-113	Pratt, Vaughan R. A Near-optimal Method for Reasoning about Action, September 1978
TM-114	Dennis, Jack B., Samuel H. Fuller, William B. Ackerman, Richard J. Swan and Kung-Song Weng Research Directions in Computer Architecture, September 1978, AD A061-222
TM-115	Bryant, Randal E. and Jack B. Dennis Concurrent Programming, October 1978, AD A061-180
TM-116	Pratt, Vaughan R. Applications of Modal Logic to Programming, December 1978
TM-117	Pratt, Vaughan R. Six Lectures on Dynamic Logic, December 1978
TM-118	Borkin, Sheldon A. Data Model Equivalence, December 1978, AD A062-753
TM-119	Shamir, Adi and Richard E. Zippel On the Security of the Merkle-Hellman Cryptographic Scheme, December 1978, AD A063-104
TM-120	Brock, Jarvis D. Operational Semantics of a Data Flow Language, S.M. Thesis, EE & CS Dept., December 1978, AD A062-997

TM-121	Jaffe, Jeffrey The Equivalence of R. E. Programs and Data Flow Schemes, January 1979
TM-122	Jaffe, Jeffrey Efficient Scheduling of Tasks Without Full Use of Processor Resources, January 1979
TM-123	Perry, Harold M. An Improved Proof of the Rabin-Hartmanis-Stearns Conjecture, S.M. & E.E. Thesis, EE & CS Dept., January 1979
TM-124	Toffoli, Tommaso Bicontinuous Extensions of Invertible Combinatorial Functions, January 1979, AD A063-886
TM-125	Shamir, Adi, Ronald L. Rivest and Leonard M. Adleman Mental Poker, February 1979, AD A066-331
TM-126	Meyer, Albert R. and Michael S. Paterson With What Frequency Are Apparently Intractable Problems Difficult?, February 1979
TM-127	Strazdas, Richard J. A Network Traffic Generator for Decnet, S.B. & S.M. Thesis, EE & CS Dept., March 1979
TM-128	Loui, Michael C. Minimum Register Allocation is Complete in Polynomial Space, March 1979
TM-129	Shamir, Adi On the Cryptocomplexity of Knapsack Systems, April 1979, AD A067-972
TM-130	Greif, Irene and Albert R. Meyer Specifying the Semantics of While-Programs: A Tutorial and Critique of a Paper by Hoare and Lauer, April 1979, AD A068-967
TM-131	Adleman, Leonard M. Time, Space and Randomness, April 1979
TM-132	Patil, Ramesh S. Design of a Program for Expert Diagnosis of Acid Base and Electrolyte Disturbances, May 1979

TM-133	Loui, Michael C. The Space Complexity of Two Pebble Games on Trees, May 1979
TM-134	Shamir, Adi How to Share a Secret, May 1979, AD A069-397
TM-135	Wyleczuk, Rosanne H. Timestamps and Capability-Based Protection in a Distributed Computer Facility, S.B. & S.M. Thesis, EE & CS Dept., June 1979
TM-136	Misunas, David P. Report on the Second Workshop on Data Flow Computer and Program Organization, June 1979
TM-137	Davis, Ernest and Jeffrey M. Jaffe Algorithms for Scheduling Tasks on Unrelated Processors, June 1979
TM-138	Pratt, Vaughan R. Dynamic Algebras: Examples, Constructions, Applications, July 1979
TM-139	Martin, William A. Roles, Co-Descriptors, and the Formal Representation of Quantified English Expressions (Revised May 1980), September 1979, AD A074-625
TM-140	Szolovits, Peter Artificial Intelligence and Clinical Problem Solving, September 1979
TM-141	Hammer, Michael and Dennis McLeod On Database Management System Architecture, October 1979, AD A076-417
TM-142	Lipski, Witold, Jr. On Data Bases with Incomplete Information, October 1979
TM-143	Leth, James W. An Intermediate Form for Data Flow Programs, S.M. Thesis, EE & CS Dept., November 1979
TM-144	Takagi, Akihiro Concurrent and Reliable Updates of Distributed Databases, November 1979

TM-145	Loui, Michael C. A Space Bound for One-Tape Multidimensional Turing Machines, November 1979
TM-146	Aoki, Donald J. A Machine Language Instruction Set for a Data Flow Processor, S.M. Thesis, EE & CS Dept., December 1979
TM-147	Schroeppel, Richard and Adi Shamir A $T = 0(2^{n/2})$, $S = 0(2^{n/4})$ Algorithm for Certain NP-Complete Problems, January 1980, AD A080-385
TM-148	Adleman, Leonard M. and Michael C. Loui Space-Bounded Simulation of Multitape Turing Machines, January 1980
TM-149	Pallottino, Stefano and Tommaso Toffoli An Efficient Algorithm for Determining the Length of the Longest Dead Path in an "Lifo" Branch-and-Bound Exploration Schema, January 1980, AD A079-912
TM-150	Meyer, Albert R. Ten Thousand and One Logics of Programming, February 1980
TM-151	Toffoli, Tommaso Reversible Computing, February 1980, AD A082-021
TM-152	Papadimitriou, Christos H. On the Complexity of Integer Programming, February 1980
TM-153	Papadimitriou, Christos H. Worst-Case and Probabilistic Analysis of a Geometric Location Problem, February 1980
TM-154	Karp, Richard M. and Christos H. Papadimitriou On Linear Characterizations of Combinatorial Optimization Problems, February 1980
TM-155	Atai, Alon, Richard J. Lipton, Christos H. Papadimitriou and M. Rodeh Covering Graphs by Simple Circuits, February 1980
TM-156	Meyer, Albert R. and Rohit Parikh Definability in Dynamic Logic, February 1980

TM-157	Meyer, Albert R. and Karl Winklmann On the Expressive Power of Dynamic Logic, February 1980
TM-158	Stark, Eugene W. Semaphore Primitives and Starvation-Free Mutual Exclusion, S.M. Thesis, EE & CS Dept., March 1980
TM-159	Pratt, Vaughan R. Dynamic Algebras and the Nature of Induction, March 1980
TM-160	Kanellakis, Paris C. On the Computational Complexity of Cardinality Constraints in Relational Databases, March 1980
TM-161	Lloyd, Errol L. Critical Path Scheduling of Task Systems with Resource and Processor Constraints, March 1980
TM-162	Marcum, Alan M. A Manager for Named, Permanent Objects, S.B. & S.M. Thesis, EE & CS Dept., April 1980, AD A083-491
TM-163	Meyer, Albert R. and Joseph Y. Halpern Axiomatic Definitions of Programming Languages: A Theoretical Assessment, April 1980
TM-164	Shamir, Adi The Cryptographic Security of Compact Knapsacks (Preliminary Report), April 1980, AD A084-456
TM-165	Finseth, Craig A. Theory and Practice of Text Editors or A Cookbook for an Emacs, S.B. Thesis, EE & CS Dept., May 1980
TM-166	Bryant, Randal E. Report on the Workshop on Self-Timed Systems, May 1980
TM-167	Pavelle, Richard and Michael Wester Computer Programs for Research in Gravitation and Differential Geometry, June 1980
TM-168	Greif, Irene Programs for Distributed Computing: The Calendar Application, July 1980, AD A087-357

TM-169	Burke, Glenn and David Moon LOOP Iteration Macro, (revised January 1981) July 1980, AD A087-372
TM-170	Ehrenfeucht, Andrzej, Rohit Parikh and Gregorz Rozenberg Pumping Lemmas for Regular Sets, August 1980
TM-171	Meyer, Albert R. What is a Model of the Lambda Calculus?, August 1980
TM-172	Paseman, William G. Some New Methods of Music Synthesis, S.M. Thesis, EE & CS Dept., August 1980, AD A090-130
TM-173	Hawkinson, Lowell B. XLMS: A Linguistic Memory System, September 1980, AD A090-033
TM-174	Arvind, Vinod Kathail and Keshav Pingali A Dataflow Architecture with Tagged Tokens, September 1980
TM-175	Meyer, Albert R., Daniel Weise and Michael C. Loui On Time Versus Space III, September 1980
TM-176	Seaquist, Carl R. A Semantics of Synchronization, S.M. Thesis, EE & CS Dept., September 1980, AD A091-015
TM-177	Sinha, Mukul K. TIMEPAD - A Performance Improving Synchronisation Mechanism for Distributed Systems, September 1980
TM-178	Arvind and Robert E. Thomas I-Structures: An Efficient Data Type for Functional Languages, September 1980
TM-179	Halpern Joseph Y. and Albert R. Meyer Axiomatic Definitions of Programming Languages, II, October 1980
TM-180	Papadimitriou, Christos H. A Theorem in Database Concurrency Control, October 1980
TM-181	Lipski, Witold Jr. and Christos H. Papadimitriou A Fast Algorithm for Testing for Safety and Detecting Deadlocks in Locked Transaction Systems, October 1980

TM-182	Itai, Alon, Christos H. Papadimitriou and Jayme Luiz Szwarefiter Hamilton Paths in Grid Graphs, October 1980
TM-183	Meyer, Albert R. A Note on the Length of Craig's Interpolants, October 1980
TM-184	Lieberman, Henry and Carl Hewitt A Real Time Garbage Collector that can Recover Temporary Storage Quickly, October 1980
TM-185	Kung, Hsing-Tsung and Christos H. Papadimitriou An Optimality Theory of Concurrency Control for Databases, November 1980, AD A092-625
TM-186	Szolovits, Peter and William A. Martin BRAND X Manual, November 1980, AD A093-041
TM-187	Fischer, Michael J., Albert R. Meyer and Michael S. Paterson CapOmega()(n log n) Lower Bounds on Length of Boolean Formulas, November 1980
TM-188	Mayr, Ernst An Effective Representation of the Reachability Set of Persistent Petri Nets, January 1981
TM-189	Mayr, Ernst Persistence of Vector Replacement Systems is Decidable, January 1981
TM-190	Ben-Ari, Mordechai, Joseph Y. Halpern and Amir Pnueli Deterministic Propositional Dynamic Logic: Finite Models, Complexity, and Completeness, January 1981.
TM-191	Parikh, Rohit Propositional Dynamic Logics of Programs: A Survey, January 198l.
TM-192	Meyer, Albert R., Robert S. Streett and Grazina Mirkowska The Deducibility Problem in Propositional Dynamic Logic, February 1981
TM-193	Yannakakis, Mihalis and Christos H. Papadimitriou Algebraic Dependencies, February 1981

TM-194	Barendregt, Henk and Giuseppe Longo Recursion Theoretic Operators and Morphisms on Numbered Sets, February 1981
TM-195	Barber, Gerald R. Record of the Workshop on Research in Office Semantics, February 1981
TM-196	Bhatt, Sandeep N. On Concentration and Connection Networks, S.M. Thesis, EE & CS Dept., March 1981
TM-197	Fredkin, Edward and Toffoli Thomaso Conservatie Logic, May 1981
TM-198	Halpern, Josepth and Reif, J. The Propositonal Dynamic Logic of Deterministic Well-Sructured Programs, March 1981
TM-199	Mayr, E. and Meyer, A. The Complexity of the Word Problems for Communative Semigroups and Polynomial Ideals, June 1981
TM-200	Burke, G. LSB Manual, June 1981
TM-201	Meyer, A. What is a Model of the lambda Calculus? Expanded Version, July 1981.
TM-202	Saltzer, J. H. Communication Ring Initialization without Central Control December 1981
TM-203	Bawden, A., Burke, G. and Hoffman, C. Maclisp Extensions, July 1981
TM-204	Halpern, J.Y. On the Expressive Power of Dynamic Logic, II, August 1981
TM-205	Kannon, R. Circuit-Size Lower Bounds and Non-Reducibility to Sparce Sets, Octoer 1981.
TM-206	Leiserson, C. and Pinter, R. Optimal Placement for River Routing, October 1981

TM-207	LONGO, G. Power Set Models For Lambda-Calculus: Theories, Expansions, Isomorphisms, November 1981
TM-208	Cosmadakis, S and Papadimitriou, C. The Traveling Salesman Problem with Many Visits to Few Cities, November 1981
TM-209	Johnson, D. and Papadimitriou, C. Computational Complexity and the Traveling Salesman Problem, December 1981
TM-210	Greif, I. Software for the 'Roiels" People Play, February 1982
TM-211	Meyer, A. and Tiuryn, J. A Note on Equivalences Among Logics of Programs, December 1981
TM-212	Elias, P. Minimax Optimal Universal Codeword Sets, January 1982
TM-213	Greif, I PCAL: A Personal Calendar, January 1982
TM-214	Meyer, A. and Mitchell, J. Terminations for Recursive Programs: Completeness and Axiomatic Definability, March 1982
TM-215	Leiserson, C. and Saxe J. Optimizing Synchronous Systems, March 1982
TM-216	Church, K. and Patil, R. Coping with Syntactic Ambiguity or How to Put the Block in the Box on the Table, April 1982.
TM-217	Wright, D. A File Transfer Program for a Personal Computer, April 1982
TM-218	Greif, I. Cooperative Office Work, Teleconferencing and Calendar Managment: A Collection of Papers, May 1982
TM-219	Jouannaud, J.-P., Lescanne, P and Reinig, F. Recursive Decomposition Ordering and Multiset ORrderings, June 1982
TM-220	Chu, T.-A. Circuit Analysis of Self-Times Elements for NMOS VLSI Systems, May 1982
TM-221	Leighton, F., Lepley, M. and Miller, G. Layouts fo the Shuffle-Exchange Graph Based on the Complex Plane Diagram, June 1982
TM-222	Meier zu Sieker, F. A Telex Gateway for the Internety, S.B. Thesis, Electrical Engineering Dept., May 1982

Technical Reports

TR-1[7]	Bobrow, Daniel G. Natural Language Input for a Computer Problem Solving System, Ph.D. Dissertation, Math. Dept., September 1964, AD 604-730
TR-2	Raphael, Bertram SIR: A Computer Program for Semantic Information Retrieval, Ph.D. Dissertation, Math. Dept., June 1964, AD 608-499
TR-3	Corbato, Fernando J. System Requirements for Multiple-Access, Time-Shared Computers, May 1964, AD 608-501
TR-4	Ross, Douglas T. and Clarence G. Feldman Verbal and Graphical Language for the AED System: A Progress Report, May 1964, AD 604-678
TR-6	Biggs, John M. and Robert D. Logcher STRESS: A Problem-Oriented Language for Structural Engineering, May 1964, AD 604-679
TR-7	Weizenbaum, Joseph OPL-1: An Open Ended Programming System within CTSS, April 1964, AD 604-680
TR-8	Greenberger, Martin The OPS-1 Manual, May 1964, AD 604-681
TR-11	Dennis, Jack B. Program Structure in a Multi-Access Computer, May 1964, AD 608-500
TR-12	Fano, Robert M. The MAC System: A Progress Report, October 1964, AD 609-296
TR-13	Greenberger, Martin A New Methodology for Computer Simulation, October 1964, AD 609-288

[7] Trs 5, 9, 10, 15 were never issued

TR-14	Roos, Daniel Use of CTSS in a Teaching Environment. November 1964, AD 661-807
TR-16	Saltzer, Jerome H. CTSS Technical Notes, March 1965, AD 612-702
TR-17	Samuel, Arthur L. Time-Sharing on a Multiconsole Computer, March 1965, AD 462-158
TR-18	Scherr, Allan Lee An Analysis of Time-Shared Computer Systems, Ph.D. Dissertation, EE Dept., June 1965, AD 470-715
TR-19	Russo, Francis John A Heuristic Approach to Alternate Routing in a Job Shop, S.B. & S.M. Thesis, Sloan School, June 1965, AD 474-018
TR-20	Wantman, Mayer Elihu CALCULAID: An On-Line System for Algebraic Computation and Analysis, S.M. Thesis, Sloan School, September 1965, AD 474-019
TR-21	Denning, Peter James Queueing Models for File Memory Operation, S.M. Thesis, EE Dept., October 1965, AD 624-943
TR-22	Greenberger, Martin The Priority Problem, November 1965, AD 625-728
TR-23	Dennis, Jack B. and Earl C. Van Horn Programming Semantics for Multi-programmed Computations, December 1965, AD 627-537
TR-24	Kaplow, Roy. Stephen Strong and John Brackett MAP: A System for On-Line Mathematical Analysis, January 1966, AD 476-443
TR-25	Stratton, William David Investigation of an Analog Technique to Decrease Pen-Tracking Time in Computer Displays, S.M. Thesis, EE Dept., March 1966, AD 631-396
TR-26	Cheek, Thomas Burrell

Design of a Low-Cost Character Generator for Remote Computer Displays, S.M. Thesis, EE Dept., March 1966, AD 631-269

TR-27 Edwards, Daniel James
OCAS - On-Line Cryptanalytic Aid System, S.M. Thesis, EE Dept., May 1966, AD 633-678

TR-28 Smith, Arthur Anshel
Input/Output in Time-Shared, Segmented, Multiprocessor Systems, S.M. Thesis, EE Dept., June 1966, AD 637-215

TR-29 Ivie, Evan Leon
Search Procedures Based on Measures of Relatedness between Documents, Ph.D. Dissertation, EE Dept., June 1966, AD 636-275

TR-30 Saltzer, Jerome Howard TRaffic Control in a Multiplexed Computer System, Sc.D. Thesis, EE Dept., July 1966, AD 635-966

TR-31 Smith, Donald L.
Models and Data Structures for Digital Logic Simulation, S.M. Thesis, EE Dept., August 1966, AD 637-192

TR-32 Teitelman, Warren
PILOT: A Step Toward Man-Computer Symbiosis, Ph.D. Dissertation, Math. Dept., September 1966, AD 638-446

TR-33 Norton, Lewis M, ADEPT - A Heuristic Program for Proving Theorems of Group Theory, Ph.D. Dissertation, Math. Dept., October 1966, AD 645-660

TR-34 Van Horn, Earl C., Jr.
Computer Design for Asynchronously Reproducible Multiprocessing, Ph.D. Dissertation, EE Dept., November 1966, AD 650-407

TR-35 Fenichel, Robert R.
An On-Line System for Algebraic Manipulation, Ph.D. Dissertation, Appl. Math. (Harvard), December 1966, AD 657-282

TR-36 Martin, William A.
Symbolic Mathematical Laboratory, Ph.D. Dissertation, EE Dept., January 1967, AD 657-283

TR-37 Guzman-Arenas, Adolfo
Some Aspects of Pattern Recognition by Computer, S.M. Thesis, EE Dept., February 1967, AD 656-041

TR-38	Rosenberg, Ronald C., Daniel W. Kennedy and Roger A. Humphrey A Low-Cost Output Terminal For Time-Shared Computers, March 1967, AD 662-027
TR-39	Forte, Allen Syntax-Based Analytic Reading of Musical Scores, April 1967, AD 661-806
TR-40	Miller, James R. On-Line Analysis for Social Scientists, May 1967, AD 668-009
TR-41	Coons, Steven A. Surfaces for Computer-Aided Design of Space Forms, June 1967, AD 663-504
TR-42	Liu, Chung L., Gabriel D. Chang and Richard E. Marks Design and Implementation of a Table-Driven Compiler System, July 1967, AD 668-960
TR-43	Wilde, Daniel U. Program Analysis by Digital Computer, Ph.D. Dissertation, EE Dept., August 1967, AD 662-224
TR-44	Gorry, G. Anthony A System for Computer-Aided Diagnosis, Ph.D. Dissertation, Sloan School, September 1967, AD 662-665
TR-45	Leal-Cantu, Nestor On the Simulation of Dynamic Systems with Lumped Parameters and Time Delays, S.M. Thesis, ME Dept., October 1967, AD 663-502
TR-46	Alsop, Joseph W. A Canonic Translator, S.B. Thesis, EE Dept., November 1967, AD 663-503

TR-47	Moses, Joel Symbolic Integration, Ph.D. Dissertation, Math. Dept., December 1967, AD 662-666
TR-48	Jones, Malcolm M. Incremental Simulation on a Time-Shared Computer, Ph.D. Dissertation, Sloan School, January 1968, AD 662-225
TR-49	Luconi, Fred L. Asynchronous Computational Structures, Ph.D. Dissertation, EE Dept., February 1968, AD 667-602
TR-50	Denning, Peter J. Resource Allocation in Multiprocess Computer Systems, Ph.D. Dissertation, EE Dept., May 1968, AD 675-554
TR-51	Charniak, Eugene CARPS, A Program which Solves Calculus Word Problems, S.M. Thesis, EE Dept., July 1968, AD 673-670
TR-52	Deitel, Harvey M. Absentee Computations in a Multiple-Access Computer System, S.M. Thesis, EE Dept., August 1968, AD 684-738
TR-53	Slutz, Donald R. The Flow Graph Schemata Model of Parallel Computation, Ph.D. Dissertation, EE Dept., September 1968, AD 683-393
TR-54	Grochow, Jerrold M. The Graphic Display as an Aid in the Monitoring of a Time-Shared Computer System, S.M. Thesis, EE Dept., October 1968, AD 689-468
TR-55	Rappaport, Robert L. Implementing Multi-Process Primitives in a Multiplexed Computer System, S.M. Thesis, EE Dept., November 1968, AD 689-469
TR-56	Thornhill, Daniel E., Robert H. Stotz, Douglas T. Ross and John E. Ward An Integrated Hardware-Software System for Computer Graphics in Time-Sharing, December 1968, AD 685-202

TR-57	Morris, James H. Lambda-Calculus Models of Programming Languages, Ph.D. Dissertation, Sloan School, December 1968, AD 683-394
TR-58	Greenbaum, Howard J. A Simulator of Multiple Interactive Users to Drive a Time-Shared Computer System, S.M. Thesis, EE Dept., January 1969, AD 686-988
TR-59	Guzman, Adolfo Computer Recognition of Three- Dimensional Objects in a Visual Scene, Ph.D. Dissertation, EE Dept., December 1968, AD 692-200
TR-60	Ledgard, Henry F. A Formal System for Defining the Syntax and Semantics of Computer Languages, Ph.D. Dissertation, EE Dept., April 1969, AD 689-305
TR-61	Baecker, Ronald M. Interactive Computer-Mediated Animation, Ph.D. Dissertation, EE Dept., June 1969, AD 690-887
TR-62	Tillman, Coyt C., Jr. EPS: An Interactive System for Solving Elliptic Boundary-Value Problems with Facilities for Data Manipulation and General-Purpose Computation, June 1969, AD 692-462
TR-63	Brackett, John W., Michael Hammer and Daniel E. Thornhill Case Study in Interactive Graphics Programming: A Circuit Drawing and Editing Program for Use with a Storage-Tube Display Terminal, October 1969, AD 699-930
TR-64	Rodriguez, Jorge E. A Graph Model for Parallel Computations, Sc.D. Thesis, EE Dept., September 1969, AD 697-759
TR-65	DeRemer, Franklin L. Practical Translators for LR(k) Languages, Ph.D. Dissertation, EE Dept., October 1969, AD 699-501
TR-66	Beyer, Wendell T. Recognition of Topological Invariants by Iterative Arrays, Ph.D. Dissertation, Math. Dept., October 1969, AD 699-502

TR-67 Vanderbilt, Dean H.
Controlled Information Sharing in a Computer Utility, Ph.D. Dissertation, EE Dept., October 1969, AD 699-503

TR-68 Selwyn, Lee L.
Economies of Scale in Computer Use: Initial Tests and Implications for The Computer Utility, Ph.D. Dissertation, Sloan School, June 1970, AD 710-011

TR-69 Gertz, Jeffrey L.
Hierarchical Associative Memories for Parallel Computation, Ph.D. Dissertation, EE Dept., June 1970, AD 711-091

TR-70 Fillat, Andrew I. and Leslie A. Kraning
Generalized Organization of Large Data-Bases: A Set-Theoretic Approach to Relations, S.B. & S.M. Thesis, EE Dept., June 1970, AD 711-060

TR-71 Fiasconaro, James G.
A Computer-Controlled Graphical Display Processor, S.M. Thesis, EE Dept., June 1970, AD 710-479

TR-72 Patil, Suhas S.
Coordination of Asynchronous Events, Sc.D. Thesis, EE Dept., June 1970, AD 711-763

TR-73 Griffith, Arnold K.
Computer Recognition of Prismatic Solids, Ph.D. Dissertation, Math. Dept., August 1970, AD 712-069

TR-74 Edelberg, Murray
Integral Convex Polyhedra and an Approach to Integralization, Ph.D. Dissertation, EE Dept., August 1970, AD 712-070

TR-75 Hebalkar, Prakash G.
Deadlock-Free Sharing of Resources in Asynchronous Systems, Sc.D. Thesis, EE Dept., September 1970, AD 713-139

TR-76 Winston, Patrick H.
Learning Structural Descriptions from Examples, Ph.D. Dissertation, EE Dept., September 1970, AD 713-988

TR-77 Haggerty, Joseph P.
Complexity Measures for Language Recognition by Canonic Systems, S.M. Thesis, EE Dept., October 1970, AD 715-134

TR-78	Madnick, Stuart E. Design Strategies for File Systems, S.M. Thesis, EE Dept. & Sloan School, October 1970, AD 714-269
TR-79	Horn, Berthold K. Shape from Shading: A Method for Obtaining the Shape of a Smooth Opaque Object from One View, Ph.D. Dissertation, EE Dept., November 1970, AD 717-336
TR-80	Clark, David D., Robert M. Graham, Jerome H. Saltzer and Michael D. Schroeder The Classroom Information and Computing Service, January 1971, AD 717-857
TR-81	Banks, Edwin R. Information Processing and Transmission in Cellular Automata, Ph.D. Dissertation, ME Dept., January 1971, AD 717-951
TR-82	Krakauer, Lawrence J. Computer Analysis of Visual Properties of Curved Objects, Ph.D. Dissertation, EE Dept., May 1971, AD 723-647
TR-83	Lewin, Donald E. In-Process Manufacturing Quality Control, Ph.D. Dissertation, Sloan School, January 1971, AD 720-098
TR-84	Winograd, Terry Procedures as a Representation for Data in a Computer Program for Understanding Natural Language, Ph.D. Dissertation, Math. Dept., February 1971, AD 721-399
TR-85	Miller, Perry L. Automatic Creation of a Code Generator from a Machine Description, E.E. Thesis, EE Dept., May 1971, AD 724-730
TR-86	Schell, Roger R. Dynamic Reconfiguration in a Modular Computer System, Ph.D. Dissertation, EE Dept., June 1971, AD 725-859
TR-87	Thomas, Robert H. A Model for Process Representation and Synthesis, Ph.D. Dissertation, EE Dept., June 1971, AD 726-049

TR-88	Welch, Terry A. Bounds on Information Retrieval Efficiency in Static File Structures, Ph.D. Dissertation, EE Dept., June 1971, AD 725-429
TR-89	Owens, Richard C., Jr. Primary Access Control in Large-Scale Time-Shared Decision Systems. S.M. Thesis, Sloan School, July 1971, AD 728-036
TR-90	Lester, Bruce P. Cost Analysis of Debugging Systems, S.B. & S.M. Thesis, EE Dept., September 1971, AD 730-521
TR-91	Smoliar, Stephen W. A Parallel Processing Model of Musical Structures, Ph.D. Dissertation, Math. Dept., September 1971, AD 731-690
TR-92	Wang, Paul S. Evaluation of Definite Integrals by Symbolic Manipulation, Ph.D. Dissertation, Math. Dept., October 1971, AD 732-005
TR-93	Greif, Irene Gloria Induction in Proofs about Programs, S.M. Thesis, EE Dept., February 1972, AD 737-701
TR-94	Hack, Michel Henri Theodore Analysis of Production Schemata by Petri Nets, S.M. Thesis, EE Dept., February 1972, AD 740-320
TR-95	Fateman, Richard J. Essays in Algebraic Simplification (A revision of a Harvard Ph.D. Dissertation), April 1972, AD 740-132
TR-96	Manning, Frank Autonomous, Synchronous Counters Constructed Only of J-K Flip-Flops, S.M. Thesis, EE Dept., May 1972, AD 744-030
TR-97	Vilfan, Bostjan The Complexity of Finite Functions, Ph.D. Dissertation, EE Dept., March 1972, AD 739-678
TR-98	Stockmeyer, Larry Joseph Bounds on Polynomial Evaluation Algorithms, S.M. Thesis, EE Dept., April 1972, AD 740-328

TR-99	Lynch, Nancy Ann Relativization of the Theory of Computational Complexity, Ph.D. Dissertation, Math. Dept., June 1972, AD 744-032
TR-100	Mandl, Robert Further Results on Hierarchies of Canonic Systems, S.M. Thesis, EE Dept., June 1972, AD 744-206
TR-101	Dennis, Jack B. On the Design and Specification of a Common Base Language, June 1972, AD 744-207
TR-102	Hossley, Robert F. Finite Tree Automata and ω-Automata, S.M. Thesis, EE Dept., September 1972, AD 749-367
TR-103	Sekino, Akira Performance Evaluation of Multiprogrammed Time-Shared Computer Systems, Ph.D. Dissertation, EE Dept., September 1972, AD 749-949
TR-104	Schroeder, Michael D. Cooperation of Mutually Suspicious Subsystems in a Computer Utility, Ph.D. Dissertation, EE Dept., September 1972, AD 750-173
TR-105	Smith, Burton J. An Analysis of Sorting Networks, Sc.D. Thesis, EE Dept., October 1972, AD 751-614
TR-106	Rackoff, Charles W. The Emptiness and Complementation Problems for Automata on Infinite Trees, S.M. Thesis, EE Dept., January 1973, AD 756<-248
TR-107	Madnick, Stuart E. Storage Hierarchy Systems, Ph.D. Dissertation, EE Dept., April 1973, AD 760-001
TR-108	Wand, Mitchell Mathematical Foundations of Formal Language Theory, Ph.D. Dissertation, Math. Dept., December 1973.
TR-109	Johnson, David S. Near-Optimal Bin Packing Algorithms, Ph.D. Dissertation, Math. Dept., June 1973, PB 222-090

TR-110	Moll, Robert Complexity Classes of Recursive Functions, Ph.D. Dissertation, Math. Dept., June 1973. AD 767-730
TR-111	Linderman, John P. Productivity in Parallel Computation Schemata, Ph.D. Dissertation, EE Dept., December 1973, PB 226-159/AS
TR-112	Hawryszkiewycz, Igor T. Semantics of Data Base Systems, Ph.D. Dissertation, EE Dept., December 1973, PB 226-061/AS
TR-113	Herrmann, Paul P. On Reducibility Among Combinatorial Problems, S.M. Thesis, Math. Dept., December 1973, PB 226-157/AS
TR-114	Metcalfe, Robert M. Packet Communication, Ph.D. Dissertation, Applied Math., Harvard University, December 1973, AD 771-430
TR-115	Rotenberg, Leo Making Computers Keep Secrets, Ph.D. Dissertation, EE Dept., February 1974, PB 229-352/AS
TR-116	Stern, Jerry A. Backup and Recovery of On-Line Information in a Computer Utility, S.M. & E.E. Thesis, EE Dept., January 1974, AD 774-141
TR-117	Clark, David D. An Input/Output Architecture for Virtual Memory Computer Systems, Ph.D. Dissertation, EE Dept., January 1974, AD 774-738
TR-118	Briabrin, Victor An Abstract Model of a Research Institute: Simple Automatic Programming Approach, March 1974, PB 231-505/AS
TR-119	Hammer, Michael M. A New Grammatical Transformation into Deterministic Top-Down Form, Ph.D. Dissertation, EE Dept., February 1974, AD 775-545
TR-120	Ramchandani, Chander Analysis of Asynchronous Concurrent Systems by Timed Petri Nets, Ph.D. Dissertation, EE Dept., February 1974, AD 775-618

TR-121	Yao, Foong F. On Lower Bounds for Selection Problems, Ph.D. Dissertation, Math. Dept., March 1974, PB 230-950/AS
TR-122	Scherf, John A. Computer and Data Security: A Comprehensive Annotated Bibliography, S.M. Thesis, Sloan School, January 1974, AD 775-546
TR-123	Introduction to Multics February 1974, AD 918-562
TR-124	Laventhal, Mark S. Verification of Programs Operating on Structured Data, S.B. & S.M. Thesis, EE Dept., March 1974, PB 231-365/AS
TR-125	Mark, William S. A Model-Debugging System, S.B. & S.M. Thesis, EE Dept., April 1974, AD 778-688
TR-126	Altman, Vernon E. A Language Implementation System, S.B. & S.M. Thesis, Sloan School, May 1974, AD 780-672
TR-127	Greenberg, Bernard S. An Experimental Analysis of Program Reference Patterns in the Multics Virtual Memory, S.M. Thesis, EE Dept., May 1974, AD 780-407
TR-128	Frankston, Robert M. The Computer Utility as a Marketplace for Computer Services, S.M. & E.E. Thesis, EE Dept., May 1974, AD 780-436
TR-129	Weissberg, Richard W. Using Interactive Graphics in Simulating the Hospital Emergency Room, S.M. Thesis, EE Dept., May 1974, AD 780-437
TR-130	Ruth, Gregory R. Analysis of Algorithm Implementations, Ph.D. Dissertation, EE Dept., May 1974, AD 780-408
TR-131	Levin, Michael Mathematical Logic for Computer Scientists, June 1974.

TR-132 Janson, Philippe A.
Removing the Dynamic Linker from the Security Kernel of a Computing Utility, S.M. Thesis, EE Dept., June 1974, AD 781-305

TR-133 Stockmeyer, Larry J.
The Complexity of Decision Problems in Automata Theory and Logic, Ph.D. Dissertation, EE Dept., July 1974, PB 235-283/AS

TR-134 Ellis, David J.
Semantics of Data Structures and References, S.M. & E.E. Thesis, EE Dept., August 1974, PB 236-594/AS

TR-135 Pfister, Gregory F.
The Computer Control of Changing Pictures, Ph.D. Dissertation, EE Dept., September 1974, AD 787-795

TR-136 Ward, Stephen A.
Functional Domains of Applicative Languages, Ph.D. Dissertation, EE Dept., September 1974, AD 787-796

TR-137 Seiferas, Joel I.
Nondeterministic Time and Space Complexity Classes, Ph.D. Dissertation, Math. Dept., September 1974.
PB 236-777/AS

TR-138 Yun, David Y. Y.
The Hensel Lemma in Algebraic Manipulation, Ph.D. Dissertation, Math. Dept., November 1974, AD A002-737

TR-139 Ferrante, Jeanne
Some Upper and Lower Bounds on Decision Procedures in Logic, Ph.D. Dissertation, Math. Dept., November 1974.
PB 238-121/AS

TR-140 Redell, David D.
Naming and Protection in Extendible Operating Systems, Ph.D. Dissertation, EE Dept., November 1974, AD A001-721

TR-141 Richards, Martin, A. Evans and R. Mabee
The BCPL Reference Manual, December 1974, AD A003-599

TR-142 Brown, Gretchen P.
Some Problems in German to English Machine Translation, S.M. & E.E. Thesis, EE Dept., December 1974, AD A003-002

TR-143	Silverman, Howard A Digitalis Therapy Advisor, S.M. Thesis, EE Dept., January 1975.
TR-144	Rackoff, Charles The Computational Complexity of Some Logical Theories, Ph.D. Dissertation, EE Dept., February 1975.
TR-145	Henderson, D. Austin The Binding Model: A Semantic Base for Modular Programming Systems, Ph.D. Dissertation, EE Dept., February 1975, AD A006-961
TR-146	Malhotra, Ashok Design Criteria for a Knowledge-Based English Language System for Management: An Experimental Analysis, Ph.D. Dissertation, EE Dept., February 1975.
TR-147	Van De Vanter, Michael L. A Formalization and Correctness Proof of the CGOL Language System, S.M. Thesis, EE Dept., March 1975.
TR-148	Johnson, Jerry Program Restructuring for Virtual Memory Systems, Ph.D. Dissertation, EE Dept., March 1975, AD A009-218
TR-149	Snyder, Alan A Portable Compiler for the Language C, S.B. & S.M. Thesis, EE Dept., May 1975, AD A010-218
TR-150	Rumbaugh, James E. A Parallel Asynchronous Computer Architecture for Data Flow Programs, Ph.D. Dissertation, EE Dept., May 1975, AD A010-918
TR-151	Manning, Frank B. Automatic Test, Configuration, and Repair of Cellular Arrays, Ph.D. Dissertation, EE Dept., June 1975, AD A012-822
TR-152	Qualitz, Joseph E. Equivalence Problems for Monadic Schemas, Ph.D. Dissertation, EE Dept., June 1975, AD A012-823
TR-153	Miller, Peter B. Strategy Selection in Medical Diagnosis, S.M. Thesis, EE & CS Dept., September 1975.

TR-154	Greif, Irene Semantics of Communicating Parallel Processes, Ph.D. Dissertation, EE & CS Dept., September 1975, AD A016-302
TR-155	Kahn, Kenneth M. Mechanization of Temporal Knowledge. S.M. Thesis, EE & CS Dept., September 1975.
TR-156	Bratt, Richard G. Minimizing the Naming Facilities Requiring Protection in a Computer Utility, S.M. Thesis, EE & CS Dept., September 1975.
TR-157	Meldman, Jeffrey A. A Preliminary Study in Computer-Aided Legal Analysis, Ph.D. Dissertation, EE & CS Dept., November 1975, AD A018-997
TR-158	Grossman, Richard W. Some Data-base Applications of Constraint Expressions, S.M. Thesis, EE & CS Dept., February 1976, AD A024-149
TR-159	Hack, Michel Petri Net Languages, March 1976.
TR-160	Bosyj, Michael A Program for the Design of Procurement Systems, S.M. Thesis, EE & CS Dept., May 1976, AD A026-688
TR-161	Hack, Michel Decidability Questions, Ph.D. Dissertation, EE & CS Dept., June 1976.
TR-162	Kent, Stephen T. Encryption-Based Protection Protocols for Interactive User-Computer Communication, S.M. Thesis, EE & CS Dept., June 1976, AD A026-911
TR-163	Montgomery, Warren A. A Secure and Flexible Model of Process Initiation for a Computer Utility, S.M. & E.E. Thesis, EE & CS Dept., June 1976.
TR-164	Reed, David P. Processor Multiplexing in a Layered Operating System, S.M. Thesis, EE & CS Dept., July 1976.
TR-165	McLeod, Dennis J.

	High Level Expression of Semantic Integrity Specifications in a Relational Data Base System, S.M. Thesis, EE & CS Dept., September 1976, AD A034-184
TR-166	Chan, Arvola Y. Index Selection in a Self-Adaptive Relational Data Base Management System, S.M. Thesis, EE & CS Dept., September 1976, AD A034-185
TR-167	Janson, Philippe A. Using Type Extension to Organize Virtual Memory Mechanisms, Ph.D. Dissertation, EE & CS Dept., September 1976.
TR-168	Pratt, Vaughan R. Semantical Considerations on Floyd-Hoare Logic, September 1976.
TR-169	Safran, Charles, James F. Desforges and Philip N. Tsichlis Diagnostic Planning and Cancer Management, September 1976.
TR-170	Furtek, Frederick C. The Logic of Systems, Ph.D. Dissertation, EE & CS Dept., December 1976.
TR-171	Huber, Andrew R. A Multi-Process Design of a Paging System, S.M. & E.E. Thesis, EE & CS Dept., December 1976.
TR-172	Mark, William S. The Reformulation Model of Expertise, Ph.D. Dissertation, EE & CS Dept., December 1976, AD A035-397
TR-173	Goodman, Nathan Coordination of Parallel Processes in the Actor Model of Computation, S.M. Thesis, EE & CS Dept., December 1976.
TR-174	Hunt, Douglas H. A Case Study of Intermodule Dependencies in a Virtual Memory Subsystem, S.M. & E.E. Thesis, EE & CS Dept., December 1976.
TR-175	Goldberg, Harold J. A Robust Environment for Program Development, S.M. Thesis, EE & CS Dept., February 1977.
TR-176	Swartout, William R.

	A Digitalis Therapy Advisor with Explanations, S.M. Thesis, EE & CS Dept., February 1977.
TR-177	Mason, Andrew H. A Layered Virtual Memory Manager, S.M. & E.E. Thesis, EE & CS Dept., May 1977.
TR-178	Bishop, Peter B. Computer Systems with a Very Large Address Space and Garbage Collection, Ph.D. Dissertation, EE & CS Dept., May 1977, AD A040-601
TR-179	Karger, Paul A. Non-Discretionary Access Control for Decentralized Computing Systems, S.M. Thesis, EE & CS Dept., May 1977, AD A040-804
TR-180	Luniewski, Allen W. A Simple and Flexible System Initialization Mechanism, S.M. & E.E. Thesis, EE & CS Dept., May 1977.
TR-181	Mayr, Ernst W. The Complexity of the Finite Containment Problem for Petri Nets, S.M. Thesis, EE & CS Dept., June 1977 .
TR-182	Brown, Gretchen P. A Framework for Processing Dialogue, June 1977, AD A042-370
TR-183	Jaffe, Jeffrey M. Semilinear Sets and Applications, S.M. Thesis, EE & CS Dept., July 1977.
TR-184	Levine, Paul H. Facilitating Interprocess Communication in a Heterogeneous Network Environment, S.B. & S.M. Thesis, EE & CS Dept., July 1977, AD A043-901
TR-185	Goldman, Barry Deadlock Detection in Computer Networks, S.B. & S.M. Thesis, EE & CS Dept., September 1977, AD A047-025
TR-186	Ackerman, William B. A Structure Memory for Data Flow Computers, S.M. Thesis, EE & CS Dept., September 1977, AD A047-026
TR-187	Long, William J.

A Program Writer, Ph.D. Dissertation, EE & CS Dept., November 1977, AD A047-595

TR-188 Bryant, Randal E.
Simulation of Packet Communication Architecture Computer Systems, S.M. Thesis, EE & CS Dept., November 1977, AD A048-290

TR-189 Ellis, David J.
Formal Specifications for Packet Communication Systems, Ph.D. Dissertation, EE & CS Dept., November 1977, AD A048-980

TR-190 Moss, J. Eliot B.
Abstract Data Types in Stack Based Languages, S.M. Thesis, EE & CS Dept., February 1978, AD A052-332

TR-191 Yonezawa, Akinori
Specification and Verification Techniques for Parallel Programs Based on Message Passing Semantics, Ph.D. Dissertation, EE & CS Dept., January 1978, AD A051-149

TR-192 Niamir, Bahram
Attribute Partitioning in a Self-Adaptive Relational Database System, S.M. Thesis, EE & CS Dept., January 1978, AD A053-292

TR-193 Schaffert, J. Craig
A Formal Definition of CLU, S.M. Thesis, EE & CS Dept., January 1978

TR-194 Hewitt, Carl and Henry Baker, Jr.
Actors and Continuous Functionals, February 1978, AD A052-266

TR-195 Bruss, Anna R.
On Time-Space Classes and Their Relation to the Theory of Real Addition, S.M. Thesis, EE & CS Dept., March 1978

TR-196 Schroeder, Michael D., David D. Clark, Jerome H. Saltzer and Douglas H. Wells
Final Report of the Multics Kernel Design Project, March 1978

[illegible]7 Baker, Henry Jr.
Actor Systems for Real-Time Computation, Ph.D. Dissertation, EE & CS Dept., March 1978, AD A053-328

TR-198	Halstead, Robert H., Jr. Multiple-Processor Implementation of Message-Passing Systems, S.M. Thesis, EE & CS Dept., April 1978, AD A054-009
TR-199	Terman, Christopher J. The Specification of Code Generation Algorithms, S.M. Thesis, EE & CS Dept., April 1978, AD A054-301
TR-200	Harel, David Logics of Programs: Axiomatics and Descriptive Power, Ph.D. Dissertation, EE & CS Dept., May 1978
TR-201	Scheifler, Robert W. A Denotational Semantics of CLU, S.M. Thesis, EE & CS Dept., June 1978
TR-202	Principato, Robert N., Jr. A Formalization of the State Machine Specification Technique, S.M. & E.E. Thesis, EE & CS Dept., July 1978
TR-203	Laventhal, Mark S. Synthesis of Synchronization Code for Data Abstractions, Ph.D. Dissertation, EE & CS Dept., July 1978, AD A058-232
TR-204	Teixeira, Thomas J. Real-Time Control Structures for Block Diagram Schemata, S.M. Thesis, EE & CS Dept., August 1978, AD A061-122
TR-205	Reed, David P. Naming and Synchronization in a Decentralized Computer System, Ph.D. Dissertation, EE & CS Dept., October 1978, AD A061-407
TR-206	Borkin, Sheldon A. Equivalence Properties of Semantic Data Models for Database Systems, Ph.D. Dissertation, EE & CS Dept., January 1979, AD A066-386
TR-207	Montgomery, Warren A. Robust Concurrency Control for a Distributed Information System, Ph.D. Dissertation, EE & CS Dept., January 1979, AD A066-996

TR-208	Krizan, Brock C. A Minicomputer Network Simulation System, S.B. & S.M. Thesis, EE & CS Dept., February 1979
TR-209	Snyder, Alan A Machine Architecture to Support an Object-Oriented Language, Ph.D. Dissertation, EE & CS Dept., March 1979, AD A068-111
TR-210	Papadimitriou, Christos H. Serializability of Concurrent Database Updates, March 1979
TR-211	Bloom, Toby Synchronization Mechanisms for Modular Programming Languages, S.M. Thesis, EE & CS Dept., April 1979, AD A069-819
TR-212	Rabin, Michael O. Digitalized Signatures and Public-Key Functions as Intractable as Factorization, March 1979
TR-213	Rabin, Michael O. Probabilistic Algorithms in Finite Fields, March 1979
TR-214	McLeod, Dennis A Semantic Data Base Model and Its Associated Structured User Interface, Ph.D. Dissertation, EE & CS Dept., March 1979, AD A068-112
TR-215	Svobodova, Liba, Barbara Liskov and David Clark Distributed Computer Systems: Structure and Semantics, April 1979, AD A070-286
TR-216	Myers, John M. Analysis of the SIMPLE Code for Dataflow Computation, June 1979
TR-217	Brown, Donna J. Storage and Access Costs for Implementations of Variable - Length Lists, Ph.D. Dissertation, EE & CS Dept., June 1979
TR-218	Ackerman, William B. and Jack B. Dennis VAL--A Value-Oriented Algorithmic Language: Preliminary Reference Manual, June 1979, AD A072-394
TR-219	Sollins, Karen R.

	Copying Complex Structures in a Distributed System, S.M. Thesis, EE & CS Dept., July 1979, AD A072-441
TR-220	Kosinski, Paul R. Denotational Semantics of Determinate and Non-Determinate Data Flow Programs, Ph.D. Dissertation, EE & CS Dept., July 1979
TR-221	Berzins, Valdis A. Abstract Model Specifications for Data Abstractions, Ph.D. Dissertation, EE & CS Dept., July 1979
TR-222	Halstead, Robert H., Jr. Reference Tree Networks: Virtual Machine and Implementation, Ph.D. Dissertation, EE & CS Dept., September 1979, AD A076-570
TR-223	Brown, Gretchen P. Toward a Computational Theory of Indirect Speech Acts, October 1979, AD A077-065
TR-224	Isaman, David L. Data-Structuring Operations in Concurrent Computations, Ph.D. Dissertation, EE & CS Dept., October 1979
TR-225	Liskov, Barbara, Russ Atkinson, Toby Bloom, Eliot Moss, Craig Schaffert, Bob Scheifler and Alan Snyder CLU Reference Manual, October 1979, AD A077-018
TR-226	Reuveni, Asher The Event Based Language and Its Multiple Processor Implementations, Ph.D. Dissertation, EE & CS Dept., January 1980, AD A081-950
TR-227	Rosenberg, Ronni L. Incomprehensible Computer Systems: Knowledge Without Wisdom, S.M. Thesis, EE & CS Dept., January 1980
TR-228	Weng, Kung-Song An Abstract Implementation for a Generalized Data Flow Language, Ph.D. Dissertation, EE & CS Dept., January 1980

TR-229	Atkinson, Russell R. Automatic Verification of Serializers, Ph.D. Dissertation, EE & CS Dept., March 1980, AD A082-885
TR-230	Baratz, Alan E. The Complexity of the Maximum Network Flow Problem, S.M. Thesis, EE & CS Dept., March 1980
TR-231	Jaffe, Jeffrey M. Parallel Computation: Synchronization, Scheduling, and Schemes, Ph.D. Dissertation, EE & CS Dept., March 1980
TR-232	Luniewski, Allen W. The Architecture of an Object Based Personal Computer, Ph.D. Dissertation, EE & CS Dept., March 1980, AD A083-433
TR-233	Kaiser, Gail E. Automatic Extension of an Augmented Transition Network Grammar for Morse Code Conversations, S.B. Thesis, EE & CS Dept., April 1980, AD A084-411
TR-234	Herlihy, Maurice P. TRansmitting Abstract Values in Messages, S.M. Thesis, EE & CS Dept., May 1980, AD A086-984
TR-235	Levin, Leonid A. A Concept of Independence with Applications in Various Fields of Mathematics, May 1980
TR-236	Lloyd, Errol L. Scheduling Task Systems with Resources, Ph.D. Dissertation, EE & CS Dept., May 1980
TR-237	Kapur, Deepak Towards a Theory for Abstract Data Types, Ph.D. Dissertation, EE & CS Dept., June 1980, AD A085-877
TR-238	Bloniarz, Peter A. The Complexity of Monotone Boolean Functions and an Algorithm for Finding Shortest Paths in a Graph, Ph.D. Dissertation, EE & CS Dept., June 1980
TR-239	Baker, Clark M. Artwork Analysis Tools for VLSI Circuits, S.M. & E.E. Thesis, EE & CS Dept., June 1980, AD A087-040

TR-240	Montz, Lynn B. Safety and Optimization Transformations for Data Flow Programs, S.M. Thesis, EE & CS Dept., July 1980
TR-241	Archer, Rowland F., Jr. Representation and Analysis of Real-Time Control Structures, S.M. Thesis, EE & CS Dept., August 1980, AD A089-828
TR-242	Loui, Michael C. Simulations Among Multidimensional Turing Machines, Ph.D. Dissertation, EE & CS Dept., August 1980
TR-243	Svobodova, Liba Management of Object Histories in the Swallow Repository, August 1980, AD A089-836
TR-244	Ruth, Gregory R. Data Driven Loops, August 1980
TR-245	Church, Kenneth W. On Memory Limitations in Natural Language Processing, S.M. Thesis, EE & CS Dept., September 1980
TR-246	Tiuryn, Jerzy A Survey of the Logic of Effective Definitions, October 1980
TR-247	Weihl, William E. Interprocedural Data Flow Analysis in the Presence of Pointers, Procedure Variables, and Label Variables, S.B.& S.M.Thesis, EE & CS Dept., October 1980
TR-248	LaPaugh, Andrea S. Algorithms for Integrated Circuit Layout: An Analytic Approach, Ph.D.Dissertation, EE & CS Dept., November 1980
TR-249	Turkle, Sherry Computers and People: Personal Computation, December 1980
TR-250	Leung, Clement Kin Cho Fault Tolerance in Packet Communication Computer Architectures, Ph.D. Dissertation, EE & CS Dept., December 1980

TR-251	Swartout, William R. Producing Explanations and Justifications of Expert Consulting Programs, Ph.D. Dissertation, EE & CS Dept., January 1981
TR-252	Arens, Gail C. Recovery of the Swallow Repository, S.M. Thesis, EE & CS Dept., January 1981, AD A096-374
TR-253	Ilson, Richard An Integrated Approach to Formatted Document Production, S.M. Thesis, EE & CS Dept., February 1981
TR-254	Ruth, Gregory, Steve Alter and William Martin A Very High Level Language for Business Data Processing, March 1981
TR-255	Kent, Stephen T. Protecting Externally. Supplied Software in Small Computers, Ph.D. Dissertation, EE & CS Dept., March 1981
TR-256	Faust, Gregory G. Semiautomatic Translation of COBOL into HIBOL, S.M. Thesis, EE & CS Dept., April 1981
TR-257	Cisari, C. Applicatin of Data Flow Architecture to Computer Music Synthesis, S.B./S.M. Thesis, EE& CS Dept., February 1981
TR-258	Singh, N. A Design Methodology for Self-Timed Systems, S.M. Thesis, EE & CS Dept., Feburary 1981

Progress Reports

Project MAC Progress Report I, to July 1964
AD 465-088

Project MAC Progress Report II, July 1964-July 1965
AD 629-494

Project MAC Progress Report III, July 1965-July 1966
AD 648-346

Project Mac Progress Report IV, July 1966-July 1967
AD 681-342

Project MAC Progress Report V, July 1967-July 1968
AD 687-770

Project MAC Progress Report VI, July 1968-July 1969
AD 705-434

Project MAC Progress Report VII, July 1969-July 1970
AD 732-767

Project MAC Progress Report VIII, July 1970-July 1971
AD 735-148

Project MAC Progress Report IX, July 1971-July 1972
AD 756-689

Project MAC Progress Report X, July 1972-July 1973
AD 771-428

Project MAC Progress Report XI, July 1973-July 1974
AD A004-966

Laboratory for Computer Science Progress Report XII,
July 1974-July 1975, AD A024-527

Laboratory for Computer Science Progress Report XIII,
July 1975-July 1976, AD A061-246

Laboratory for Computer Science Progress Report XIV,
July 1976-July 1977, AD A061-932

Laboratory for Computer Science Progress Report 15,
July 1977-July 1978, AD A073-958

Laboratory for Computer Science Progress Report 16,
July 1978-July 1979, AD A088-355

Laboratory for Computer Science Progress Report 17,
July 1979-July 1980, AD A093-384

Laboratory for Computer Science Progress Report 18,
July 1980-June 1981