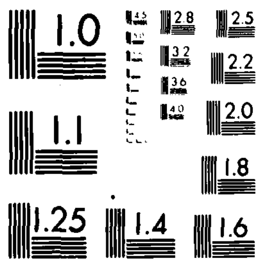
AD-A088 355 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/6 9/2
LABORATORY FOR COMPUTER SCIENCE PROGRESS REPORT 16, 1 JULY 1978--ETC(U)
AUG 80 M L DERTOUZOS N00014-75-C-0661
UNCLASSIFIED LCS/PR-16 NL

1 of 2

1 OF 2

AD-
A088355



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

12

LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

AO73958

Progress Report 16

July 1978 - June 1979

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
LCS Progress Report 16 ✓

2. GOVT ACCESSION NO.
AD-A088

3. RECIPIENT'S CATALOG NUMBER
355

4. TITLE (and Subtitle)
6
Laboratory for Computer Science
Progress Report 16,
1 July 1978 - June 1979
9

5. TYPE OF REPORT & PERIOD COVERED
DARPA-DOD
Progress Report 7/78-6/79

6. PERFORMING ORG. REPORT NUMBER
14 LCS/PR-16

7. AUTHOR(s)
Laboratory for Computer Science Participants
M. L. Dertouzos, Director

8. CONTRACT OR GRANT NUMBER(s)
15 N00014-75-C-0661 ✓

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Laboratory for Computer Science ✓
Massachusetts Institute of Technology
545 Technology Square, Cambridge, Ma. 02139 11

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Defense Advanced Research Projects Agency 1
1400 Wilson Blvd.
Arlington, Va. 22209

12. REPORT DATE
Augu[redacted] 1980

13. NUMBER OF PAGES
194

14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*
Office of Naval Research
Department of the Navy
Information Systems Program
Arlington, Va. 22217
12 196

15. SECURITY CLASS. *(of this report)*
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*
10 Michael L./Dertouzos

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

18. SUPPLEMENTARY NOTES

automata theory
CLU
complexity
database systems
distributed systems
dynamic modeling
natural language
network protocols
on-line computers

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

computation structures
computer languages
computer networks
computer systems
data intensive planning
information systems
knowledge-based systems
local networks
Morse-code
multi-access computers
personal computers
planning systems
programming languages
real-time computers

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

This report summarizes the research performed by the MIT Laboratory for Computer Science from July 1, 1978 to June 30, 1979.

401 641

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

1/B

LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

Progress Report 16

July 1978 - June 1979

Prepared for the

Defense Advanced Research Projects Agency

A

Effective date of contract:	1 January 1975
Contract expiration date:	31 December 1980
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. DARPA Order No. 2095

Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government or any other person or agency connected with them.

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Work reported herein was carried out within the Laboratory for Computer Science, an MIT interdepartmental laboratory. During 1978-1979 the principal financial support (55%) of the Laboratory has come from the Defense Advanced Research Projects Agency (DARPA), under Office of Naval Research Contract N00014-75-C-0661. DARPA has been instrumental in supporting most of our research during the last 16 years and is gratefully acknowledged here.

TABLE OF CONTENTS

ADMINISTRATION

Academic Staff

M. L. Dertouzos	Director
A. R. Meyer	Associate Director

Administrative Staff

M. E. Baker	Administrative Assistant
P. G. Heinmiller	Librarian
E. I. Kampits	Administrative Officer
T. E. Lightburn	Fiscal Officer
G. L. Wallace	Purchasing Agent

Support Staff

G. W. Brown
L. S. Cavallaro
C. Cornish
M. J. Cummings
S. Geitz

J. Jones
D. Kontrimus
M. Profirio
T. Sealy
P. Vancini

INTRODUCTION

This annual report to the Defense Advanced Research Projects Agency (DARPA) describes research performed at the MIT Laboratory for Computer Science (formerly Project MAC), funded by that agency and monitored by the Office of Naval Research during the period July 1, 1978--June 30, 1979.

The Laboratory for Computer Science 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 knowledge in programs that comprehend typed natural-language (English) queries, and the use of knowledge about programming by an automated "programming apprentice."

Research in the second 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 the programming of real-time systems that control physical processes from higher-level, domain-specific languages. Other research examples in this area include the study of data bases, the architecture of individual "personal" machines, and the networking and organization of geographically distributed systems of computers.

The Laboratory's third principal area of research, Theory, involves exploration and development of theoretical foundations in computer science and is sponsored primarily by NSF.

The fourth area of Laboratory research is entitled Computers and People and entails societal as well as technical aspects of the interrelationships between people and machines. This area is sponsored primarily by industrial organizations.

During the past year, the Laboratory consisted of 279 members--36 faculty, 4 visiting faculty, 17 visitors, 90 professional and support staff, 85 graduate and 47 undergraduate students--organized into 14 research groups. During the reporting period, Professor Albert Meyer was appointed Associate Director of the Laboratory, replacing Professor Joel Moses, who became Associate Head for Computer Science and Engineering in the Department of Electrical Engineering and Computer Science. The academic affiliation of most of the faculty and students is with the Department of Electrical Engineering and Computer Science.

The 1978-79 year was very active. Technical results were disseminated through the publications of the Laboratory members and will not be discussed here. Highlights of the year included the following.

The Technical Services group formed in 1978 now functions in both a research and support role within LCS. The group is responsible for the development and implementation of hardware for the LCS local network, which is part of our distributed systems effort. Its support functions include development and maintenance of an electronics laboratory facility as well as management and maintenance of the LCS network. During the year, the group achieved operation of the local LCS network.

Since 1977, geographically distributed systems have evolved into a major Laboratory focus, involving about half of our Laboratory personnel and resources. Research in this area strives to make possible geographically distributed systems consisting of a large number of processors. The central theme of our research involves local autonomy of each processor, as well as application cohesiveness of the overall system. This theme is pursued at the various levels of representation that characterize this research. In particular, at the hardware level, the Real Time Systems group is developing a single-user computer that will be manufactured for us by the Heath Company, while the Technical Services group is pursuing the network that will link at least 100 of these machines within our Laboratory.

At the operating system level, the Real Time Systems group is pursuing research in and development of a distributed operating system that will reside on these machines. The Computer Systems Research and Programming Methodology groups are pursuing a general-purpose language especially suited to the semantics of distributed systems. At the applications level, our newly formed Office Automation group is researching the semantics of a language suitable for describing office procedures. At the same level, our Programming Technology group is researching the structure of a system that makes possible planning in the presence of large amounts of data.

Another newly formed research group, Digital Information Mechanics, under Professor Edward Fredkin, focuses its research on a new approach for implementing fast digital circuits. In this approach, energy representing bits of information is spatially steered, rather than consumed, as is the case with today's circuits.

The Laboratory's Distinguished Lecturer Series, initiated in 1976, has proved very successful in attracting members of the MIT community. The 1978-79 lecturers under this series were Ruth M. Davis (Deputy Under Secretary of Defense for Research and Engineering, Department of Defense), Lewis M. Branscomb (Vice President and Chief Scientist, IBM), Juris Hartmanis (Professor and Chairman, Department of Computer Science, Cornell University), Butler W. Lampson (Senior Research Fellow, Xerox Corp.), Alan J. Perlis (Eugene Higgins Professor of Computer Science, Yale University), Herbert A. Simon (Professor of Computer Science and Psychology, Carnegie-Mellon University). Professor Simon's presentation on "Learning by Examples and Learning by Doing," was given to an overflow audience at MIT on November 2, 1978, one week after his being awarded the Nobel Prize in economics.

During 1978-79, research in previously established areas yielded several new results which were published through Laboratory technical reports (TR202-TR218) and technical memoranda (TM106-TM137), as well as through articles in the technical literature.

Michael L. Dertouzos
Director

Assembly and final editing of this report was done by J. Badal, V. E. Golden and Dr. E. I. Kampits.

COMPUTER SYSTEMS RESEARCH GROUP

Academic Staff

J. H. Saltzer, Groupleader
D. D. Clark
F. J. Corbato
I. Greif
D. P. Reed
L. Svobodova

Research Staff

J. N. Chiappa
E. A. Martin

Undergraduate Students

R. Baldwin
D. Bollinger
H. Carter
C. Davis
D. Gorman
R. Gorman
C. Hornig
K. Khalsa
R. Lawhorn
G. Simpson
S. Szymanski
S. Toner

Graduate Students

W. Ames
G. Arens
E. Ciccarelli
W. Gramlich
M. Herlihy
S. Kent
V. Ketelboeter
A. Luniewski
A. Marcum
A. Mendelsohn
W. Montgomery
K. Sollins
R. Wyleczuk

Support Staff

R. Bisbee
G. Chambers
J. Jones
M. Webber

Visitors

A. Takagi

COMPUTER SYSTEMS RESEARCH GROUP

A. INTRODUCTION

Three LCS groups, Computer Systems Research, Programming Methodology, and Technical Services, are working closely together on a joint project to create a new kind of distributed programming environment. This environment involves software and hardware for a local ring network, internetwork interconnection, the personal desktop computer being designed by the DSSR Group, specialized service-providing computers, and finally design and implementation of programming language extensions that make the overall distributed environment easy to apply. Some distributed applications are also being developed, to provide additional guidance and feedback on the utility of the underlying system. The primary distinguishing feature of this research project is its rationale for distribution of function: the project assumes that the dominant force that determines where function will be distributed is administrative autonomy. Last year's progress report provides arguments for this assumption. Descriptions of the various parts of this project will be found in the individual progress reports of the three groups.

The Computer Systems Research part of this joint project this year involved four aspects:

1. Development of semantics for distributed applications;
2. Network and internetwork software design and implementation;
3. Specialized server design;
4. Experimental distributed application development.

These four aspects are discussed in the next four sections.

B. PROGRAMMING FOR DISTRIBUTED APPLICATIONS

This year, part of our group has concentrated on the problem of performing an update that involves several physical nodes. While not all distributed applications will require such rigorous control as is implied by the protocols that have emerged from this body of work, mechanisms for performing distributed updates atomically belong among the basic mechanisms of a distributed operating system. Traditional approaches to coordination and synchronization based on semaphores, locks, path expressions, or message passing do not seem to provide the guidance one might hope, because they do not include reliability and recovery mechanisms. In the distributed environment, errors and error recovery apparently must be considered explicitly as part of every mechanism and protocol including coordination of parallel activities. Since separating the problem into coordination and recovery seems not to work, other lines of separation must be sought. One line of separation that appears to have some promise is to work separately on consistency and atomicity. The idea is to on the one hand develop strategies for assuring consistency of multi-site data for single transactions run with no interference from other transactions (but with the possibility of failure) and on the other hand develop general techniques for insuring atomicity of multi-site operations in the face of possible failures, so that no transaction ever sees internal states of others.

This line of separation has been explored in depth and has provided several insights and advances. Three separate research reports by Reed [11], Montgomery [10] and Takagi [14] present several innovative approaches and mechanisms. An analysis of the recent results and a summary of insights that have emerged are presented in a report by Svobodova [13].

Coordination of concurrent processes is difficult in a distributed system because of communication delays and modularity. In a centralized system with shared memory, coordination can be achieved inexpensively by locking the data to be accessed while the computation uses it. Locking is inexpensive, because all processes can easily access the locks, and because deadlock detection or avoidance can be centralized. In a distributed system, locking requires interactions among the users of the data and therefore imposes communications delays. Furthermore, deadlock detection is impractical because it requires global knowledge of all computations and their locks. Deadlock avoidance is impractical because a module of a distributed computation that uses modules at other nodes may not have knowledge of the data accesses or the order of access at those nodes.

Recovery from failures is made difficult in a distributed system by the peculiar nature of communication failures. In particular, when node A requires a service from node B that involves modifying data objects stored at node B, certain kinds of communication failures will leave node A in doubt as to whether node B has performed the requested action or not. The requesting computation at node A has only one option at this point, since further actions by node A are usually contingent upon successful completion of the request at B to insure consistency between various parts of the system. Node A must wait until node B's state can be ascertained, but this may take a very long time. If node A holds resources needed by other computations, then such a failure can cause deadlock.

In a monolithic distributed data base, such failures may be tolerable, since each node and communication link is maintained to a high standard of availability. In a system where nodes are autonomously managed, such failures are more likely to happen, and more likely to be of long duration. For example, after node A sends its request, but before B responds, node B (a desktop computer) may be powered off for lunch.

1. Atomic Actions

The goal of this research is to support the construction of atomic actions. An atomic action is an operation on data whose effects on data are completely specified by the algorithm executed by the atomic action. In particular, though the atomic action may access (read or update) many pieces of data, each many times, as part of its execution, the effect of the atomic action can be described as a relation between the initial state of all of the data items it touches and their final states when the atomic action is finished.

Atomic actions require both synchronization and recovery mechanisms in their accesses to data. Synchronization is required to ensure that no other computations within the system can observe an intermediate state of the data objects accessed. If an intermediate state of an object could be observed outside the atomic action then the behavior of the atomic action could not be specified solely in terms of a relation between initial and final states of the objects accessed. Synchronization is required to ensure also that no other computation can modify any data object used by the atomic action during its execution. That is, the atomic action's program can be written without any consideration of interference form concurrent access to the data it accesses. Recovery mechanisms are required to ensure that if a failure occurs, preventing completion of an atomic action, the intermediate state of the data resulting from partial completion of an atomic action is not exposed to observation by other computations.

Our concept of atomic actions is quite similar to that of Lomet [8] and also similar to the sphere of control described by Davies [2, 3]. If all computations in the systems perform all their data accesses as part of atomic actions, then the observable behavior of the system will be the same as a serial schedule, as in the definition of atomic transaction developed by Eswaran, et al. [4].

The simplest implementation of atomic actions is to delay all other computations in the system for the duration of the atomic action. This is often inefficient in a single processor system, but in a distributed system connected by a network, it may be impossible, because of communications failures.

It is sufficient, however, to guarantee that an atomic action has exclusive access to the data it actually reads or updates. Locking is often used to achieve this exclusiveness, by associating a lock (or mutual exclusion semaphore) with each data object that will be used by a computation before that computation can access the data. Locking introduces the possibility of deadlock, the detection of which may be quite difficult in a distributed system, while classic deadlock avoidance techniques cannot cope with transactions whose data accesses are unknown, due to the presence of information-hiding mechanisms that hide the representations of objects, or due to the use of pointers or accesses otherwise predicated on values obtained earlier in an atomic action's execution.

The essence of locking is to seize exclusive access to a group of objects for a period of time. Thus, the proper behavior of an atomic action is controlled indirectly, by ensuring that the timing of its steps is properly coordinated with the timing of other computations. The basis of the locking approach to implementing atomic actions is that there is one instant or interval during the atomic action at which all locks are simultaneously held. That interval must either precede or follow the corresponding interval of any potentially interfering atomic action.

In contrast, the mechanism proposed by Reed, and extended by Takagi, coordinates the access to a set of objects by a naming mechanism that gives names to a sequence of versions (virtual global states) of the system. Actions on each object specify the particular version to be affected. There are two naming mechanisms described below. Psuedo-times are a totally ordered set of names referring to successive virtual states of the system's data. Possibilities are a mechanism for referring to groups of updates to objects for the purpose of error recovery.

Atomic actions are implemented by giving the virtual processor executing the atomic action exclusive use of both a sequence of psuedo-times, derived from the real time at which the action begins, and a possibility. Access to a particular object in a particular state of the system requires that both a possibility and a psuedo-time for that state of the system be used as parameters to the access. There is a very close analogy between this approach to implementing atomic actions and the capability approach to protection of data [1, 5]. In both approaches, having a name for something is a prerequisite for its use, so exclusive use can be granted by restricting the propagation of names.

A major result of Reed's approach is that atomic actions are modularly composable operations. That is, one can implement atomic actions so that new atomic actions can be constructed out of previously existing atomic actions without either (a) modifying the preexisting implementations or (b) requiring that the new actions know what objects the preexisting atomic actions access. Locking mechanisms for providing synchronization or recovery for atomic actions make it difficult thus to compose atomic actions because of the need to have at least one instant of time where all data touched by an atomic action is locked. Composing atomic actions in a system based on locking thus requires extending the time during which an object is locked. To ensure atomicity, updates of a distributed database are coordinated by a two-phase commit protocol [6, 7]. The problem of how to schedule actions that are part of different (concurrent) operations is resolved during the first phase. In this phase, the individual participants can proceed independently. In the second phase, a careful coordination of all participants is necessary to ensure that either all of them commit or all of them abort their part of the operation. A particularly simple form of two-phase commit protocol is central to the implementation of Reed's possibilities [11].

a. Scheduling Actions on Distributed Objects

Our approaches differ from the traditional use of locking to schedule actions on multiple objects. Instead of locking, the key idea is that at each object the actions must take effect in the proper order, achieving the proper ordering can be done independently at each object, however. The mechanisms of Reed and Takagi use the ordered sequence of versions to record the proper order of the reads and updates applied to an object. Montgomery, on the other hand, uses an atomic broadcast protocol to ensure that object accesses always arrive in the proper order; out of order requests do not occur in his scheme [10].[1]

The mechanisms of Reed and Takagi vary in how they handle out of order (outdated) updates. One approach is to discard a delayed (older) update (and consequently the operation that generated that request) if a new read (that is, a request with a higher timestamp) has already been processed [11]. However, this may lead to a "dynamic deadlock" where the same set of operations is aborted over and over because those operations repeatedly outdate each other - this is similar to the collision problem in contention networks such as Ethernet [9]. A different approach is to discard a newer request in favor of a delayed older request, given that the operation that generated this newer request has not yet been committed [14]. This solution may lead to starvation (i.e., a specific operation may never succeed since other operations will always cause it to abort), but it is free of dynamic deadlock. An extension is to allow multiple versions of objects to coexist: a delayed read request can be satisfied without having to discard any other request if the particular version still exists [11, 14].

b. Cascading of Backout

It is interesting to analyze how atomicity can be ensured without requiring that the objects updated by an operation not be available to other operations until after the final decision regarding the commitment of the first operation. Allowing early release of uncommitted information leads to the problem of cascading of backout should the final decision be "abort". To be able to handle such a cascaded backout, two conditions have to be satisfied:

1. During a backout of each individual operation, the operation does not need to "reacquire" (in an exclusive mode) any of the objects that it has read or modified in order to undo the changes;

2. It is possible to remember (or to reconstruct) all information flow among concurrent operations.

Since concurrent operations do not know of one another and their dependencies, it is difficult to properly backout a set of dependent operations if the recovery data is maintained by individual operations. Thus the recovery schemes ought to be object-oriented. Object-oriented recovery means that all the information needed to restore an object to some previous value if associated with the object (provided by the manager of the object) rather than with the individual operations.

[1] This arrival ordering has an interesting application to the case of replicated data. Read requests are guaranteed to arrive at a copy only after any preceding write copy updates have been compelted at or on this copy, even if other copies have not yet been updated.

To provide an object oriented recovery that allows the new value of the object to be seen before the end of the second phase of the operation that generated this new value, one can implement "multiple uncommitted versions" of the object. A new version of an object is created when the manager of the object receives the new value for the object; this action does not destroy the old value of the object (that is, the old version). A version represents the (possible) state of the object. In addition to having a value, a version has a time attribute that specifies its range of validity. The range of validity of a particular version is the time interval in the history of the object during which the object was in the state represented by that version. A version is only tentative until the operation that created it is committed. If the operation fails, the version is simply discarded. If a version is discarded, that part of the object history is erased. Now if another operation can read an uncommitted version Vx and create its own version Vy of the same or another object such that the value or even the existency of Vy depends on Vx, it is necessary to remember that Vy is dependent on Vx, since if Vx is discarded, Vy must be discarded also.

Multiple uncommitted versions were used by both Takagi and Montgomery. In Montgomery's scheme, a request to read an object that currently has several uncommitted versions will return a set of all possible values that the outstanding (not yet committed) operations could produce. This set is called a polyvalue. Thus, when an object is made visible but before the operation that modified it is completed, both the old and the new value (each of which themselves may already be a polyvalue because the outcome of some earlier operation has not yet been resolved) are presented to the next operation. If this next operation needs a precise answer, it will have to wait. If it is sufficient to know that all values possible as of that time are within an acceptable range, the operation can calculate new values for each polyvalue component, and if the answer is satisfactory, it may even commit. In this scheme, if one operation is aborted, no other operations ever have to be backed out; the only thing that has to be done is to throw away some irrelevant information, thus reducing the polyvalue set. In this sense, the scheme is symmetric for the two possible outcomes of an operation - this pruning has to be done both for the commit and the abort decision. That is, the scheme does not make any assumption about the probability of success. More important, it allows operations to be committed before the earlier operations that modified the same objects have been commited (or aborted). Takagi's scheme is more conservative. Here an operation cannot commit until all the operations on which it depends have committed. If any such earlier operation is aborted, all dependent operations must be backed out. Takagi assumes that failures are rare, that is, once a new version is created, it is very likely that it will be committed; put in different words, with high probability it is the right value that the next operation should see. Thus, a read request returns the value of the newest version.

2. Summary

Although significant progress has been made towards understanding the role of atomic operations in a distributed system and the mechanisms required to implement them, more careful thought is still needed. Among the issues that need further investigation are:

a. Defining atomic operations as operations on abstract objects;

b. Atomic operation on data that has been replicated to achieve higher availability;

c. The relation between requirements for atomic transactions is distributed systems and the requirements for interruptibility in processor instruction set design.

Based on the studies of the various proposed schemes and, in particular, of the assumptions underlying those schemes, it seems that some of the approaches might be overly conservative and unbalanced in their relative emphasis on different classes of problems. An essential step towards making a significant progress in this area is to get a better understanding of the importance, frequency and severity of the specific problems that the individual schemes for atomic updates attempt to solve.

3. Other Work on Semantics for Distributed Applications

Sollins' thesis [12] presents a model of a distributed system in which the universe of objects in the distributed system is divided into mutually exclusive sets, each set corresponding to a context. This model allows naming beyond the context boundaries, but limits communication across such boundaries to message passing only. A number of activities require the ability to copy objects, for example parameter passing across context boundaries and providing greater reliability through redundancy. Therefore, copying of complex data structures is investigated in this model, and semantics, algorithms, and sample implementations are presented for three candidate copy operations. Two important goals in the development of the copy operations were (1) to allow the contexts autonomy in naming the objects contained within them and (2) to reflect as much as possible the structure of an object in a copy of it. A new type of object, the message-context was developed to achieve these by recording the names of components and providing a translation into names not local to the context.

CLU provides two kinds of copy operations, copy1, which copies only the top level of a complex structure, and copy, which copies the complete structure. The definition of copy does not allow for discovery of components contained more than once unless the programmer includes a procedure for detecting sharing in his own implementation of copy operations. In addition, neither of these operations reflects the model in which the context boundary plays any part. For this reason, in addition to providing the copy1 and copy (renamed copy-one and copy-full for clarity) with modifications to achieve our goals, a new operation copy-full-local is introduced. This operation reflects the nature of the model by copying complex structures to the boundaries of the context containing the object, but not beyond. The reason this operation is different from copy-full is that containment of components can span more than one context, because naming can occur across context boundaries, although communication across such boundaries is limited to message passing. Because communication with foreign contexts may be difficult or impossible, the copy-full-local operation may provide the most complete copying possible.

C. NETWORK AND INTERNETWORK SOFTWARE DESIGN AND IMPLEMENTATION

A great deal of effort has been invested this year in the planning, coordination, and implementation of protocols for our local network. Much of this work is reported in detail in internal memoranda (Network Implementation Notes No. 9, 10, 11, and 12). Highlights are mentioned here.

This year saw the stabilization and implementation of an ARPA-standard internet protocol, version four of the Transmission Control Protocol (TCP). CSR group members participated actively in the discussion that led to this newly stabilized design. In addition, we implemented TCP for the Multics System. Implementations of TCP (and its relatives) have been done elsewhere for the UNIX system and TOPS-20. These developments make use of TCP feasible as the primary protocol for our local net. The UNIX version may be transportable to the VAX, locally leaving only ITS without a minimum-effort implementation for TCP. Participation in the ARPANET TCP development and local implementation has involved a substantial amount of time for several members of our group, a cost that we hope will abate in the coming year.

Along with software for the various host machines, we have procured software and hardware for a variety of special servers to be connected to our Local Net. Perhaps most important is the gateway between the Local Net and the ARPANET. The hardware for this machine, a Digital Equipment Corporation LSI-11, has been purchased, and software for this machine is now under development. Much of the software was imported from SRI International, and the portion that must be written here is completely designed and partially written. We hope that it will be running within a month.

Another special server for the Local Net is a Terminal Interface Unit (TIU) which will allow the connection of single display terminals directly to the Local Net, rather than to a particular host machine. Again, we have procured an LSI-11 to run this server, and imported the basic TIU software from SRI. However, many local enhancements are required to this software, including the implementation of a driver for the Local Net Interface and the locally popular protocol for controlling a process from a remote terminal, SUPDUP. These enhancements are currently under way.

We have developed a new protocol, called Trivial File Transfer Protocol or TFTP, that is easier to implement than the ARPANET File Transfer Protocol. By implementing this protocol first, when adding a new machine to the Local Net, it becomes immediately possible to move files, including other protocol programs implemented elsewhere, onto the machine very early in the development of its network software. TFTP has been implemented and tested on UNIX, and files have been transferred over the net using TFTP between UNIX machines. It is being implemented for Multics, VAX, and TOPS-20. TFTP thus provides a route by which programs for VAX VMS can be transferred onto that machine, a current pressing need within the Laboratory.

In summary, the status of the Local Net is as follows. As reported elsewhere, the hardware has been operational for several months. A large quantity of software is on the verge of working, and much of it has already gone through a preliminary stage of debugging. We hope that in the next few months, perhaps by the end of the summer, that there will be a substantial number of useful services actually available over the Local Net. The status of software for the Local Net is being reported in a series of internal memoranda, to which the interested reader is referred.

1. Specialized Servers

As discussed above, machines other than hosts are connected to the Local Net in order to provide particular network services. The services currently under development are rather low level: gateways and terminal controllers. At the same time, we are making preliminary plans for more sophisticated sorts of service, which we hope to develop during the next year. The two most interesting servers are a file server and an authentication server. The file server is very important, as more and more small machines are installed at the Laboratory. A local disk may double the cost of the small machine, so economics alone suggests that a clustered file system is a very important facility on our net. At the same time, construction of a file server allows us to explore several ideas having to do with management of remote data in a reliable consistent manner. As mentioned in section A, above, several schemes for distributed management of data have proposed by members of our group. Designing and implementing a file server with the update semantics proposed by Reed in his thesis will give us valuable experience in the area.

In support of a file server, and other servers as well, it is necessary to have some server that authenticates a particular request, so that information stored on a remote file server is not completely unprotected. We expect this server to be a separate logical entity, which will take on the general task of confirming the identity of the requester of a service. Preliminary design for such a server is currently under way. The extreme importance of such a server becomes clear when one examines the list of other services which might possibly be connected to the Local Net. For example, it has been suggested that it might be possible to connect an airline reservation service to the Local Net, so that one could obtain airline tickets from ones terminal. Clearly, we cannot allow uncontrolled purchasing of airline tickets from within the Laboratory. Thus, inclusion of such a service, which seems a very real part of an office environment, places a strong requirement for an authentication server that we can truly trust.

Other servers that we have considered for the Local Net include specialized printing machines, most particularly a laser printer, access to various institute data bases such as an online version of the telephone book, access to the Official Airline Guide, which is less dangerous than the actual ability to make reservations, access to other message sending facilities such as Telex or TWIX, a WWV accurate time source, and some sort of low cost archival storage, perhaps done by connecting the Datacomputer directly to the Local Net, or by making use of archival storage that may possibly be purchased for the M.I.T. IBM system/370. Exploring the possibilities of other servers will be done as ideas and resources permit.

D. APPLICATIONS FOR DISTRIBUTED SYSTEMS

Research projects involving the building of applications for distributed systems can vary a great deal in character and may appear in many groups here - from office automation to artificial intelligence. We plan to coordinate the building of several applications within the distributed systems group in an effort to study the systems issues common to a variety of applications and particularly to organize the feedback into the design of extended CLU.

Currently, the only application system being built is a calendar system. The calendar system will be built as a set of active forms. One can think of making an appointment as the filling in of a form, namely the calendar. However, filling in the form may cause actions, such as the sending of messages to other calendars. The act of filling in a form may take place during more than one interactive session, and may involve mail in cases where some calendars are either unavailable or are unable to make commitments. Emphasis is on semi-public calendars such as the schedule for the conference rooms in the building, so that we can expect the system to be used by a group of people without the overhead on putting many personal calendars online.

This approach will result in a calendar system that contrasts with others that have been built on top of shared data bases that contain information from all participating calendars. In keeping with an assumption of local autonomy and in order to mirror the existing organization of offices in which people keep their own calendars privately, we must assume that calendars (personal or for public resources) may be stored on personal computers and therefore may not always be available. This will lead to the investigation of a different set of systems issues, such as facilitating the buffering of requests to calendars which are unavailable and providing for resumptions of conversations when participants are available.

The notion of active forms should be applicable to many systems, and is particularly natural as a model for communication in office automation systems. Thus attempts will be made to use this approach in the design of applications in an effort to develop a unified approach to application design and implementation.

An immediate result of the initial design phase of the calendar has been the development of a distributed programming environment on the XX TOPS-20 time-sharing system. The environment is built on top of CLU and relies on a network interface built by David Reed. Currently, we can send and receive both CLU objects and datagrams used for internet communication. Work is in progress on implementing typed ports to facilitate compile time type checking of messages. There is a student working on implementing stable storage of CLU objects, a project that will be useful also in development of a file server. Other facilities to be implemented are guardians and a port server. We are following the design of extended CLU and are trying to simulate intended features whenever possible. However, the design of this environment is intended to be driven by application requirements and may incorporate features with which we would like to experiment even if they are not of such obvious generality as to be included in the design of extended CLU.

The design of applications is also involved with the project on development of special servers. As mentioned above, the implementation for stable storage has already appeared as a meeting point for both projects. Requirements for backup storage and processing as well as for authentication are further examples of needs that can be met by work in either or both projects. By proceeding with applications and systems development simultaneously, we will be providing immediate users of the servers' facilities.

References

1. Dennis, J., and Van Horn, E. "Programming semantics for multiprogrammed computations," Communications ACM 22, 3 (March 1966), 143-153.

2. Davies, C. T. "Recovery semantics for a DB/DC system," Proceedings 1973 ACM National Conference, Atlanta, Ga., 1973, 136-141.

3. Davies, C. T. "Data processing spheres of control," IBM Systems Journal 17, 2 (1978), 179-198.

4. Eswaran, K. et al. "The notions of consistency and predicate locking in a database system," Communications ACM 19, 11 (November 1976), 624-633.

5. Fabry, R. "Capability-based addressing," Communications ACM 17, 7 (July 1974), 403-412.

6. Gray, J. "Notes on data base operating systems," Lecture Notes in Computer Science, 60, Springer-Verlag, Berlin, 1978, 393-481.

7. Lampson, B., et al. "Crash recovery in a distributed data storage system," Xerox PARC, Palo Alto, Calif., 1976, to appear in Communications ACM.

8. Lomet, D. "Process structuring, synchronization, and recovery using atomic actions," Proceedings ACM Conference on Language Design for Reliable Software, Raleigh, N.C., March 1977. Also in ACM SIGPLAN Notices 12, 3 (March 1977), 128-137.

9. Metcalfe, R. "Ethernet: Distributed packet switching for local computer networks," Communications ACM 19, 7 (July 1976), 395-404.

10. Montgomery, W. "Robust concurrency control for a distributed information system," MIT/LCS/TR-207, MIT, Laboratory for Computer Science, Cambridge, Ma., December 1978.

11. Reed, D. "Naming and synchronization in a decentralized computer system," MIT/LCS/TR-205, MIT, Laboratory for Computer Science, Cambridge, Ma., September 1978.

12. Sollins, K. "Copying complex structures in a distributed system," MIT/LCS/TR-219, MIT, Laboratory for Computer Science, Cambridge, Ma., May 1979.

13. Svobodova, L. "Building reliable distributed systems: the problem of atomic operations," Request for Comments No. 173, MIT, Laboratory for Computer Science, Computer Systems Research Group, Cambridge, Ma., May 1979.

14. Takagi, A. "Concurrent and reliable updates of distributed databases," Request for Comments No. 167, MIT, Laboratory for Computer Science, Computer Systems Research Group, Cambridge, Ma., November 1978.

Publications

1. Clark, D., Pogran, K., and Reed, D. "An introduction to local area networks," Proceedings of the IEEE 66, 11 (November 1978), 1497-1517.

2. Corbato, F., and Clingen, C. "A managerial view of the multics system development," in Research Directions in Software Technology, Peter Wegner (Ed.), MIT Press, Cambridge, Ma., 1979, 139-158.

3. Greif, I., and Meyer, A. "Specifying programming language semantics," Conference Record of the Sixth Annual ACM Symposium on Principles of Programming Languages, San Antonio, Tx., January 1979, 180-189.

4. Greif, I., and Meyer, A. "Specifying the semantics of while-programs: A tutorial and critique of a paper by Hoare and Lauer," MIT/LCS/TM-130, MIT, Laboratory for Computer Science, Cambridge, Ma., April 1979. (Submitted for journal publication.)

5. Kent, S. "Protocol design considerations for network security," Proceedings of the NATO Advanced Studies Institute on Interlinking of Computer Networks, Bonas, France, August 1978.

6. Kent, S. "Privacy and security in networks," in Protocols and Techniques for Data Communication Networks, Franklin Kuo (Ed.), Prentice-Hall, Englewood Cliffs, N.J., to be published April 1980.

7. Kent, S. "A comparison of some aspects of public-key and conventional cryptosystems," Proceedings International Conference on Communications, Boston, Ma., June 1979.

8. Reed, D. "Using naming for synchronizing access to decentralized data," ACM Seventh Symposium on Operating Systems Principles, Pacific Grove, Calif., December 1979, 163.

9. Reed, D., and Kanodia, R. "Synchronization with eventcounts and sequencers," Communications ACM 22, 2 (February 1979), 115-123.

10. Saltzer, J. "Performance analysis and evaluation: no connection with reality," Research Directions in Software Technology, Peter Wegner (Ed.), MIT Press, Cambridge, Ma., 1979, 652-654.

11. Saltzer, J., Pogran, K. "A star-shaped ring network with high maintainability," Proceedings NBS-Mitre Local Area Communications Network Symposium, May 1979.

12. Svobodova, L. "Performance problems in distributed systems," INFOR (Canadian Journal of Operational Research and Information Processing), 1978.

13. Svobodova, L. "Reliability issues in distributed information processing systems," Proceedings of 9th International Symposium on Fault Tolerant Computing, Madison, Wis., June 1979, 9-16.

14. Svobodova, L., Liskov, B., Clark, D. "Distributed computer systems: structure and semantics," MIT/LCS/TR-215, MIT, Laboratory for Computer Science, Cambridge, Ma., March 1979.

Theses Completed

1. Bollinger, D. "A computer controlled telephone interface," B.S. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., May 1979.

2. Lamson, R. "An EMACS interface to Multics objects," B.S. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., May 1979.

3. Marcum, A. "A manager for named, permanent objects," M.S. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., June 1979.

4. Montgomery, W. "Robust concurrency control for a distributed information system," MIT/LCS/TR-207, Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., November 1978.

5. Nevins, R. "An efficient logic simulator for the trident guidance computer," M.S. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., December 1979.

6. Reed, D. "Naming and synchronization in a decentralized computer system," MIT/LCS/TR-205, Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., September 1978.

7. Simmons, S. "Comparison of microcomputers dedicated hardware systems in communications," B.S. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., October 1978.

8. Sollins, K. "Copying complex structures in a distributed system," MIT/LCS/TR-219, M.S. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., May 1979.

9. Strazdas, R. "A network traffic generator for DECNET," MIT/LCS/TR-127, M.S. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., January 1979.

10. Woltman, G. "Controlling terminals with high-level protocols," M.S. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., August 1978.

11. Wyleczuk, R. "Timestamps and capability-based protection in a distributed data base environment," MIT/LCS/TM-135, M.S. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., February 1979.

Theses in Progress

1. Ames, W. "A local area network simulator," M.S. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected February 1980.

2. Hornig, C. "A second generation network interface for Multics," B.S. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected, August 1979.

3. Luniewski, A. "The architecture of an object based personal computer," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected February, 1980.

4. Toner, S. "Dynamic message routing in interconnected local area data networks," B.S. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected August 1979.

Talks

1. Clark, D. and Pogran, K. "Local networks," GTE Sylvania, Needham, Ma., December 1978.

2. Kent, S. "Secure data communications and storage standards," NBS Invitational Workshop on Audit and Evaluation of Computer Security II, Mi., Florida, November 1978.

3. Kent, S. "Network security," Sandia Laboratories, Albuquerque, N.M., March 1979.

4. Kent, S. "Implementing protected subsystems in decentralized computing environments," Bell Northern Research and SRI International, Palo Alto, Ca., May 1979.

5. Kent, S. "Comparisons of conventional and public-key cryptosystems," Session on Network Security at the National Computer Conference, New York, N.Y., June 1979.

6. Reed, D. "Implementing modular atomic actions," IBM San Jose Research Laboratory, San Jose, Ca., January 1979.

7. Reed, D. "Naming and synchronization in a decentralized computer system," Ford Motor Company Research Division, Dearborn, Mi., December 1978.

8. Saltzer, J. "The impact of changing technology on security," Workshop on Computer Security, Mitre Corporation, Bedford, Ma., February 1979.

9. Saltzer, J. "Thoughts on system structure - the impact on changing technology," Series of 7 lectures given at Tata Institute of Fundamental Research, Bombay, India, May 1979.

10. Saltzer, J. IRIA 2nd International Conference on Operating Systems, Program Committee Member and Session Chairman, Rocquencourt, France, October 2-5, 1978.

11. Svobodova, L. "LCS research in the area of distributed computing," and "Structure of distributed computer systems," IBM, San Jose Research Laboratory, San Jose, Ca., August 1978.

12. Svobodova, L. "Design and operation of distributed computer system," IBM, Thomas J. Watson Research Center, Yorktown Heights, N.Y., October 1978.

13. Svobodova, L. "Distributed systems: structure and semantics," Bell Laboratories, Murray Hill, N.J., April 1979.

Committee Memberships

Chiappa, Noel. DARPA IPTO Internet TCP Working Group.

Clark, David. DARPA IPTO Internet TCP Working Group.

Reed, David. DARPA IPTO Internet TCP Working Group.

Saltzer, Jerome. Draper Laboratory Committee on 1979 Summer Security Workshop.

Saltzer, Jerome. DoD/DDRE Security Working Group Member.

DATABASE SYSTEMS GROUP

Academic Staff

M. Hammer, Group Leader

Graduate Students

B. Berkowitz
E. Cardoza
A. Chan
D. McLeod
B. Niamir
S. Sarin
S. Zdonik

Support Staff

M. Martinez
M. Nieuwkerk

DATABASE SYSTEMS

A. INTRODUCTION

The principal focus of our work this year has been the continued exploitation of the Semantic Data Model (SDM), a high-level data modelling mechanism developed by our group. The SDM enables the description of a database to be couched in terms that are natural to the application environment and that are unrelated to issues of physical data storage and representation. Such an SDM schema can be used to support a number of powerful and advanced database system capabilities that can not be readily achieved in conventional systems. This year we have applied it to the issues of automatic database design, knowledge-based query processing, and improved database application programming.

B. AUTOMATIC DATA BASE DESIGN

Arvola Chan has been working on the development of a systematic methodology for automating the physical database design process in the context of an intelligent self-organizing database management system. Unlike previous studies of physical database design, which have usually assumed the simplified environment of a database that contains only one type of logical entity and which have typically focused on a single specific design issue (such as index selection or attribute partitioning), it is our premise that an automated design facility should account for the co-access to different types of information frequently exhibited by applications in an integrated database environment; furthermore, it should not be required to select its synthesis of a good physical organization from a limited repertoire. Since we have shown that even the relatively simple problem of selecting an optimal set of indices for a database consisting of a single relation in the context of a conventional relational database system (such as INGRES or System R) is np-hard, we believe it would be futile to attempt to formulate and solve a realistically complex physical database problem through the use of mathematical optimization techniques. The thrust of our work, therefore, is to utilize database semantics and heuristic search techniques to synthesize "good" physical organizations at a reasonable cost. Our methodology for designing and implementing integrated databases revolves about: (1) the use of a conceptual data model to describe the semantics of the application environment; (2) the employment of a non-procedural scheme to express the database usage characteristics of applications; and (3) the use of the database semantics by heuristic strategies in selecting a physical organization to match the prevailing access requirements. The description of the conceptual schema used by the automatic design facility is couched in terms of the Semantic Data Model, previously developed by our group and outlined in last year's report. The use of such a semantically oriented data model serves: (1) to clarify the interconnections among different types of logical entities modelled by the database, thereby facilitating efficiency oriented transformations in their representations; (2) to form the basis for the specification of fundamental semantic integrity constraints whose maintenance during database updates may have significant impact on the selection of physical organization; and (3) to provide a framework for the specification of how redundant information can be included in the database so as to enhance the range of query optimization opportunities.

To ensure the maintainability of application programs and the economic feasibility of the physical redesign process, we stipulate that the data requirements of applications be specified in non-procedural terms. We allow for an individual application to access data concerning several types of logically related entities. Furthermore, unlike conventional relational database management systems, which assume that the data required by an application is always to be formatted as a flat file, we attempt to take into consideration the data formatting requirements of applications as well (i.e., how the data is to be organized and sorted). Our premise is that these concerns are important to the selection of an efficient design. Furthermore, accounting for them reduces the computation costs of evaluating the merits of a candidate design, by limiting the number of reasonable strategies for processing a non-procedurally specified transaction that must be considered.

The design space addressed by the automatic design facility includes such dimensions as horizontal partitioning, vertical partitioning, primary file organization, index selection, parent-child chaining, hierarchical clustering, attribute migration, iterator inversion, and several others. The size and complexity of this design space preclude the possibility of finding the optimal design within it. Rather, we seek to locate, by heuristic means, a "good" design without incurring great expense in the search process. To this end we have examined heuristic techniques that have been successfully applied to simpler database design problems as well as to combinatorial optimization problems in other disciplines. Our overall heuristic search strategy is based on the notions of iterative improvement and problem decompositon. Iterative improvement involves defining a neighborhood structure around each point in the design space. Given a starting point that is taken as the incumbent design, the points within its neighborhood are examined and evaluated. One which constitutes the best improvement over the incumbent design is adopted as the new incumbent design, and the process is repeated until the incumbent design constitutes a locally optimal point within its neighborhood. In general, starting the iterative improvement process from different starting points will yield different locally optimal designs. Indeed, it might be desirable to obtain such multiple designs and to select the best one from among them. Our experience and experimentation have shown that this strategy, when coupled with a good choice for an initial design, can yield accurate results when applied to individual physical design issues, such as secondary index selection or attribute partitioning for a single logical file. However, when confronted with a multi-dimensional design space for a database that models multiple types of logical entities, the judicious choice of an initial physical organization as the basis for iterative improvement, and the appropriate definition of a neighborhood structure, are too complex for the success of such a heuristic hill-climbing approach.

Our strategy, therefore, is to break the design process down into two phases: an initial design phase and an iterative improvement phase. During the initial design phase, a divide and conquer strategy is applied. Knowledge of data semantics and data characteristics is first used to obtain a default representation for each type of logical entity in the schema. This provides a basis for comparing various strategies for decomposing those transactions in the usage pattern that involve multiple types of logical entities into simpler ones that involve only individual types of logical entities. A better representation for each type of logical entity can then be selected, based on the derived transaction decomposition schemes. Different dimensions of design freedom can then be heuristically ordered and explored in a univariate fashion (i.e., at each step of the heuristic search procedure, alterations to only one design issue will be considered.) In a sense, the purpose of the initial design phase is to obtain a good non-redundant preliminary design for each entity type, which is based purely on "local" criteria. During the iterative improvement phase, these representations for the various types of logical entity, which had been selected in a decoupled fashion, are taken as starting points for further improvement. The costs of processing each class of transactions are evaluated in detail in the context of the initial design (and subsequently in the context of any improved designs). The perturbations to the incumbent design that may be considered at this stage include duplication of attributes, consolidation of the representations of several types of logical entities, clustering of the representations of different types of logical entities, and explicit maintenance of derivable information. In other words, this phase is concerned with "inter" entity type design issues. It is also during this phase that desirable physical redundancies are introduced.

To reduce the number of possibilities at each step of the iterative improvement process, a problem decomposition strategy is employed. More specifically, we partition the logical entity types into groups, and make perturbations to the representations of one group of logical entity types at a time. This partitioning is determined heuristically in such a way that logical entity types that are often accessed together are assigned to the same group; the techniques employed are those used in graph partitioning problems and in clustering problems. Perturbations to the representations of different groups of logical entity types are considered sequentially. Again, the order in which the groups are considered is determined heuristically. (Different orderings may lead to different final designs, since some transactions may cross group boundaries.) The perturbation process is a goal-directed one. The bottlenecks in the processing of those transactions that access logical entity types within the group are identified, and a list is constructed of those changes to the physical organization that would ameliorate these bottlenecks. To improve the efficiency of the process of computing the merits of any proposed organization, an incremental evaluation strategy is used. The organizational changes preferred by different transactions are used as the candidates for the perturbation, and the improvements they would render are computed. (When the number of candidates is large, a rough estimate of the payoff for each proposed change can be used to restrict attention to the more promising ones.) When no significant improvement can be made to the group of entity types currently under consideration, attention is turned to the next group of entity types. After the representation of each group of entity types has been considered for perturbation, a new round of the iterative improvement process is performed, provided that a significant improvement was achieved during the previous round.

The details of this heuristic design procedure have largely been worked out. Our next effort is to embody this design methodology in a prototype design facility, and to experiment with it. We have performed some preliminary investigations into possible strategies for evaluating the performance of the design heuristics. Because of the complexity of our heuristics, it would be futile to look for analytic means of deriving bounds on the deviation from optimality of the designs that it produces. Instead, we will have to rely on empirical evaluation. One difficulty that confronts us is that there is no computationally feasible way to locate, and determine the cost associated with, the truly optimal design for any given benchmark problem. Hence, we will need indirect methods for measuring the performance of the design heuristics. One strategy that has been successfully employed in evaluating the performance of heuristics in combinatorial optimization problems (such as travelling salesman problems) is to use a heuristic algorithm that will generate different heuristic solutions when given different starting points. The value of the globally optimal solution can then be approximated from the distribution of the values of the heuristic solutions, by means of a statistical inference procedure. We have applied such a technique to the problem of estimating the values of globally optimal solutions for simple physical design problems (such as the problem of selecting indices or attribute partitions for a single file), and have compared the estimated values against the values found by exhaustive enumeration (for problems of small size). The preliminary results are quite encouraging, suggesting that this technique can be employed to validate other database design heuristics. Hence, we will develop a version of the basic heuristic design procedure that generates multiple designs for any given problem, and then apply statistical inference procedures to estimate the value of the best achievable design; this value can then be compared against the value of the design obtained using the basic heuristic design procedure, and the quality of the procedure thereby ascertained. In addition, we plan to compare the designs generated by our heuristic design facility with designs developed by experienced human designers.

Bahram Niamir has worked on two problems related to the automatic physical design of databases. The first was the completion of earlier work on the automatic selection of attribute partitions for a relational database. The results of this research on attribute partitioning were written up in the form of a paper for wider dissemination; the paper has been presented at a conference.

The other project completed in this period consisted of the generalization and extension of earlier results in estimating the number of block accesses in blocked database organizations. In a previous report, we reported on the derivation of the block access formula, which estimates the number of block accesses required to retrieve a set of randomly distributed records that reside on directly accessible blocks of a disk storage. Assume we have a file of n records stored b records to a block. Further assume that r records have been randomly selected (their addresses are known) and are to be retrieved from the file. We assume that the r selected records appear uniformly throughout the file, that each record has equal probability of being among the r selected records, and that no block of records will be retrieved more than once. This model of access is realistic and holds for databases where an index on an unsorted field provides direct access to the records. Assuming this access configuration, the number of block accesses to retrieve the r records is

$$A(n, b, r) = (n/b)[1 - bin(n-r, b)/bin(n, b)].$$

where bin(j, k) is the binomial coefficient.

We have extended the above formula to accommodate more generalized files. Our first extension allowed records to overlap block boundaries and span more than one block (i.e., b is non-integer.) The second extension allows the last block in the file to be only partially full (i.e., n/b is non-integer) and hence have a smaller probability of being accessed. We have derived a generalized block access formula that incorporates these two extensions. This generalized block access formula reduces to the special case mentioned above when b and n/b become integral.

The computation of the (general) block access formula entails the costly calculation of the binomial coefficient. We have derived approximations to this computation that are extremely accurate and efficient and that compare favorably to other approximations to the block access formula that have been suggested in the literature.

C. DATABASE PROGRAMMING LANGUAGE DESIGN

Brian Berkowitz has been designing a programming language for coding data intensive application systems (such as purchasing systems and accounting systems). The language is designed to integrate a database management system into a programming environment.

The conventional approach to implementing an applications system is to augment a general purpose programming language by allowing it to employ a data management system to obtain and modify values from a database. Our approach is to develop a unified and integrated facility that directly incorporates data management capabilities into a programming language and moreover includes in the languages those constructs most natural and useful for coding database application programs.

We have decided to use the Semantic Data Model as the languages data model (i.e., the mechanism used to describe the data used by an application system). The basic contents of this model are entities and classes, which are collections of entities; attributes describe members of classes. An attribute can also be used to link a member of a class to associated members of other classes. A number of other features are provided to enable precise modelling of the structure of the application domain. This data model has the advantage of allowing data bases to be constructed that model their real world counterparts more naturally than would be possible using lower level data models such as relations or networks.

The basic SDM operations are the creation of new classes and attributes. New classes are created by a variety of methods including restriction (selecting those entities in a class that satisfy some predicate), mapping class C under an attribute A (selecting entities that are associated with a member of C by attribute A), intersecting two classes, and taking the union of two classes. Queries on an SDM database can be expressed in terms of sequences of these operations. A new attribute can be created by specifying the attribute name and a way of obtaining its value. The SDM provides a variety of mechanisms for specifying how the value of an attribute is to be derived, including arithmetic expressions involving other attributes and a number of specialized functions.

We have employed two examples to aid us in developing and evaluating our language design, a purchasing system and a job scheduling system; the first is typical of transaction processing applications, the latter of batch systems. Both systems were coded in a stepwise manner, and evolved from simple starting points to fairly complex systems. The purpose of using this technique was to determine how easily systems written in our language could be modified and enhanced.

The purchasing system is an interactive system that allows requisitions for goods to be written and that controls the subsequent processing of these requisitions. Some of the operations performed by the system include obtaining approval for requisitions, aiding buyers in ordering goods for the requisitions (either from stock or from a vendor), processing invoices, and recording receipt of delivered goods.

The job scheduling system is concerned with producing a schedule for work to be done on machines in a job shop. A set of orders to be filled and a schedule of machine availability is provided as input to the system. The system uses characteristics of the machines, parts explosions of the products ordered, information about vendor deliveries, and so on, in order to produce a schedule of assemblies to be done on the machines in the shop. This example provides a complementary example to the purchasing system. It is not interactive, but rather correlates large amounts of data in order to produce a schedule.

We view a data-intensive application system as consisting of a centralized database and a set of programs that access it, where a program contains both procedures and local databases. There is a separate version of each local database for every instantiation of a program; the local database can be accessed by each procedure in the program, but is destroyed when the program terminates its execution. An example of such a local data base is a data base maintained for each buyer when he processes requisitions. On the other hand, the centralized data base can be accessed by all users of the system, it is permanent, and only one version of it exists.

Classes and entities may be declared as local variables and may be passed as procedure arguments. The SDM operations mentioned above are the primary means for manipulating these data objects. Classes passed as arguments are called local classes and can be manipulated on in the same way as classes in the centralized data base. New local classes can be created using the assignment operation. A procedure can act as a multi-valued function, returning a number of local classes or entities to the caller. Consider the following function that returns the average waiting time for delivery of a particular good.

```
Define Average_wait(Item:[a GOOD]) <returns [an INTERVAL]>
  [Good_orders <- ORDERS with Good_ordered=Item
     {Waiting_period <- Date_received - Date_ordered}
   return(Average(Good_orders.Waiting_period))]
```

This function takes a GOOD entity called ITEM as an argument. It examines the ORDERS class in the centralized data base to find the orders for Item. Good_orders is the result of this query, and is a local subclass of ORDERS. It has a new attribute defined, Waiting_period, which is the number of days between the order date and the delivery date of the order. The average of these Waiting_periods is the result of this function.

The foregoing summarizes the basic data objects and operations in the language. It is our premise that the control structures provided by a programming language play a major role in determining the level of complexity of programs written in it. Rather than merely including conventional, general purpose control structures, we decided to abstract the most common forms of algorithm structure encountered in database applications, and incorporate them in the language.

The most important iteration construct provided is the For each statement:

For each e in C S

where e is a variable, C is a class, and S is a statement. The construct is executed in the following manner: S is evaluated with e successively bound to each element in C. This construct has been extended to allow early termination conditions (e.g., where some predicate is tested either before each execution of S), and iteration over several classes at once.

The language also provides a variety of conditional statements and a sophisticated interrupt system, similar to that found in CLU. Several important specialized types of procedures are included. These procedures allow for specialized types of control that cannot be easily simulated using normal procedures. Two examples of these special procedures are:

> Case_procedures: these provide the ability to dispatch a number of different procedures based on the class of an entity provided as an argument, on the value of some expression computed from the arguments, or on a set of predicates.
>
> Iterators: these provide a mechanism for creating a class. The iterator contains a loop which in turn contains yield statements that identify entities that are to form a new class returned by the iterator. This is similar to the Iterators available in CLU.

An important issue in designing any data manipulation language is the facility for updating the centralized database. Update_procedures and entry_procedures are the only type of procedures that we allow to use operations that can update the central data base of the system. The code of one of these procedures must consist of a sequence of elementary update operations, such as adding entities to a class, deleting entities from a class, initializing an attribute of an entity, and changing an attributes value. These procedures may contain no iteration or conditional constructs. The purpose of this limitation is to restrict and simplify the ways in which the database is changed. Update_procedures are explicitly invoked by procedures in the system while entry_procedures are triggered automatically when an entity is added to a particular class.

These procedures serve as the basic building blocks for transactions, i.e., operations that change the database. A transaction maps a database from one consistent state to another consistent state; the state in the midst of a transaction need not be consistent. A transaction should appear as an atomic operation to the outside world. It is very important that if the system fails in the midst of a transaction, none of the changes specified by a transaction are visible when the system recovers. We have defined our language so that procedures reading information from the database always see a consistent view of the database; they do not see any of the changes made by transactions running concurrently with them. This view of the database will not be appropriate for transactions that are updating a class; they need to see the most recent version of that class. Therefore, we provide a mechanism that allows transactions to see the most recent version of the data that they are interested in changing.

In order to coordinate multiple users who may be interested in modifying the same data, special modules called controllers can be written. There is only one instance of each controller in the system. The controller has its own private data base, which it uses to keep track of running transactions. Normally, only one transaction can interact with the controller at a time; this serialization is used to provide synchronization. The controller has the ability to abort transactions that have invoked it, which may be necessary in order to prevent deadlock.

We are in the midst of considering several other important facilities for our language. For one, we would like to impose some global structure on programs written in our language, which would allow them to be seen as more than just a disjoint set of procedures. A facility for defining interactive conversations between the user and the system is also being developed. These interactions often require complex display and data entry capabilities. Data verification often involves checking relationships between data in several entered fields, and various recovery options should be possible in the case of detected erroneous entry.

D. KNOWLEDGE BASED QUERY PROCESSING

Stanley Zdonik has been working on the problem of how to use application-specific knowledge in the processing of data base queries. In many contemporary data base systems, users are provided with a high-level language for specifying the set of records that are to be retrieved. Such a language does not include any means for specifying how the request is to be implemented. The query processing system is responsible for making the choice of access paths that will most efficiently produce the desired set of records. Typically, this system exploits only syntactic properties of the query and information about available file structures and access paths. The premise of our work is that knowledge about the semantics of the application, provided independently of the query, can be used tυ transform the query into a more efficient form.

We view a query as an expression whose value is a set of records. The purpose of this research is to develop techniques that will produce an alternative formulation of the query that is less expensive to evaluate than the original. This process will involve the use of a collection of statements about the application environment that will be called the knowledge base. If the original query expression is Exp1, and if Exp2 is the expression constructed by applying the knowledge based query processor to Exp1, then Exp1 and Exp2 are equivalent, and Exp2 is cheaper than Exp1 to evaluate.

For example, suppose the original query were to find all employees who work in the accounting department. This might be expressed in the following way:

restriction [Employees, lambda (e) dept [e] = 'accounting']

Here, we are asking for those members of the set Employees that satisfy the condition expressed by the lambda expression, that is, those employees whose dept attribute is equal to the value 'accounting'. If the system knows the additional facts that the accounting employees are a subset of the clerks and the secretaries, and that the clerks and the secretaries are a subset of Employees, then we can form an equivalent request as follows:

```
restriction [
  restriction [Employees, lambda (e)
            member of [job-type [e], {clerk, secretary}]],
    lambda (e) dept [e] = 'accounting']
```

If we further know that the attribute job-type has an available index (allowing direct access to those records with a given job type value) while the attribute dept does not, it is reasonable to assume that the second query will be easier to evaluate than the first. The restriction on job-type can be rapidly evaluated, yielding a smaller set of entities that have to be sequentially searched for ones that satisfy the restriction on dept.

A system of this type should not spend a large amount of time trying to produce a cheaper equivalent for a query that is only going to be executed once. Considerations such as this have led us to the following design principles.

1. The system should be most responsive to queries that do not involve deeply nested expressions. This is the most common type of request expressed by users.
2. The number of steps in the chain of deductions that are required to produce a new query should be small. Complicated inferences should not be required to achieve the transformations in which we are interested.
3. It is not necessary that the system discover every possible transformation on the original query, or even the optimal one. It is more important that it find a reasonable transformation in most of the common cases. If it occassionally misses some possibility, this is acceptable in the interest of efficient operation of the optimization system itself.

In order to specify the knowledge base, the Data Base Administrator must use a language that provides all of the statement types that are useful for making query transformations. We believe that all of the types of query improvement that we have investigated so far can be accomplished with the use of a knowledge base expressed with a rather small set of statement types. These statement types form the DBA language and are as follows:

1. Subset of [Exp1, Exp2]. This states that the set specified by Exp1 is a subset of the set specified by Exp2.
2. Equivalent [Exp1, Exp2]. This states that the set specified by Exp1 is the same set as the set specified by Exp2.
3. Equivalent Condition [Exp, Condition1, Condition2]. This states that the result of restricting the set specified by Exp on Condition1 is the same as restricting it on Condition2.
4. Possible Values of Attribute [Exp1, Attribute, Exp2]. This states that the value of Attribute for members of the set Exp1 must be drawn from the set Exp2. For example, Possible Values of Attribute [Employees, dept, {management, accounting, production}] says that the

attribute named dept for members of the set Employees can only assume the three values given.

5. Impossible Values of Attribute [Expl, Attribute, Exp2]. This states that the value of Attribute for members of Expl must not include values from Exp2.
6. Derived Values of Attribute [Expl, Attribute, Lambda exp]. This states that the value of Attribute for each member of Expl can be derived by applying the function Lambda to the member. Lambda is a function that is defined in terms of some other attribute of the objects in Expl. For example, Derived Values of Attribute [Employees, Fica, lambda (e) 0.05*salary[e]] says that the value of an employees FICA attribute can be derived from the value of his salary by multiplying it by 0.05.

These facilities completely determine the statements that the DBA can express. Once these statements have been entered, they can be compiled into a form that can be used by the program that is performing the optimizations. (Deduction of new facts based on the ones entered by the DBA can be done at compile time too.) We have developed a number of different improving transformations and the techniques for using the knowledge base to recognize when they apply and to perform them.

We have also designed a uniform system architecture that will facilitate the design of new transformation techniques. This architecture provides a framework for structuring such new techniques. We plan to develop a prototype implementation of this system in the next year.

Publications

1. Hammer, M. and Niamir, B. "A heuristic approach to attribute partitioning," Proceedings of the SIGMOD International Conference on Management of Data, Boston, Ma., 1979.
2. Hammer, M. "Research directions in database management," in Research Directions in Software Technology, Wegner, P. (Ed.), MIT Press, Cambridge, Ma., 1979.
3. Hammer, M. and Ruth, G. "Automating the software development process," in Research Directions in Software Technology, Wegner, P. (Ed.), MIT Press, Cambridge, Ma., 1979.
4. Hammer, M. "Application oriented software research," in Research Directions in Software Technology, Wegner, P. (Ed.), MIT Press, Cambridge, Ma., 1979.
5. Chan, A. and Niamir B. "On estimating the cost of accessing records in blocked database organizations," submitted to ACM Transactions on Database Systems.
6. McLeod, D. "A semantic database model and its associated structured user interface," MIT/LCS/TR-214, MIT, Laboratory for Computer Science, Cambridge, Ma., August 1978.

Theses Completed

1. Leong, R. "Cost minimization in database integrity checking," S.B. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., August 1978.
2. Dell'Aquila, J.B. "Error detection and correction in database updates using imprecise constraint predicates," S.B. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., August 1978.
3. McLeod, D. "The semantic data base model and its associated structured user interface," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., August 1978.
4. Wang, L. "Simultaneous file partitioning and index selection in a self-adaptive data base management system," S.B. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., August 1978.

Theses in Progress

1. Chan, A. "A methodology for automating the physical design of integrated data bases," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected December 1979.
2. Koschella, J. "Some optimizations of nested data base queries, "S.B. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected August 1979.
3. Zdonik, S. "Semantic query optimization in data base systems," S.M. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected August 1979.
4. Berkowitz, B. "A programming language for data-intensive application systems," S.M. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected August 1979.
5. Sarin, S. "Reliability in distributed database systems," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected August 1980.
6. Niamir, B. "Distributed database design," Ph.D dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected August 1980.

Talks

1. Hammer, M. "A distributed data management system for C^3," ONR-MIT Workshop on Distributed Information and Decision Systems, Cambridge, Ma., August 1978.
2. Hammer, M. "Distributed database management for command and control," MIT LIDS Colloquium, Cambridge, Ma., December 1978.
3. Hammer, M. "On database management system architecture," Informal DBMS Workshop, Boston, Ma., April 1979.
4. Hammer, M. "Directions in database research," 1979 ACM SIGMOD International Conference on Management of Data, Boston, Ma., May 1979.
5. Niamir, B. "A heuristic approach to attribute partitioning," ACM SIGMOD International Conference on Management of Data, Boston, Ma., May 31, 1979.

DIGITAL INFORMATION MECHANICS GROUP

Academic Staff

E. Fredkin, Group Leader

Research Staff

T. Toffoli

Graduate Students

Robert Giansiracusa

Undergraduate Students

Andrew Ressler

Support Staff

M. Briggs

DIGITAL INFORMATION MECHANICS GROUP

A. INTRODUCTION

Discrete Information Mechanics is the name that we have chosen to give to our major area of research. Thus we will be known as the Discrete Information Mechanics Group. It should be noted that within the group there are also activities going on that belong to other areas. The main areas of research are:

A. Information Mechanics

1. Conservative Logic
2. Energy dissipation in computational processes
3. Reversible Programming Languages
4. Reversible Systems Theory
5. Reversible Difference Equations
6. Digital Mechanics

B. Semi-Intelligent Systems

7. Robotics Balance and Walking
8. Robotics Learning through Experience
9. Semi-Intelligent Control Systems

B. CONSERVATIVE LOGIC

We have continued to make progress in the area of conservative logic through several avenues. Andrew Ressler completed his senior thesis on "Practical circuits using conservative reversible logic." The main accomplishments of this thesis were the development of a particularly convenient notational scheme for the representation of conservative logic circuits, and the detailed design of efficient realizations, in the form of conservative-logic networks, of several complex functions that had been attempted before without success. These implementations are very useful insofar as they make clearer what is involved in the design of practical circuits of the complexity found in modern computers. Another student tackled the task of designing an efficient multiplier, and a great deal of work was done on the general problem of reversible multiplication. Circuits that perform efficient multiplication with conservative logic have the property that they are also good circuits for factoring large integers. In fact, by running an efficient multiplier backwards, one obtains a reasonably efficient algorithm for factoring large integers. (By "reasonably efficient" we mean about as good as the best algorithms published in Knuth.) The main emphasis of our efforts on factoring is to provide a problem area for the further development of ideas on reversibility in general.

This work is expected to continue next year, with the emphasis shifted from the "hardware" domain of conservative logic to the "software" domain, with the development of reversible programming languages (see 4 below).

C. ENERGY DISSIPATION IN COMPUTATIONAL PROCESSES

We have submitted a proposal on "Design Principal for Achieving High-Performance Submicron Digital Technologies" to DARPA. This proposal has been well received, and is almost certain to be funded within the next few months. The concept is to take advantage of the properties of conservative logic in order to design circuitry that dissipates much less power than what would result from other design methodologies. In fact, conservative-logic principles provide much lower theoretical bounds for the minimum energy that must be dissipated during the operation of computer circuits.

We will be exploring the consequences of these design principles for various advanced technologies, including CMOS, bubble circuitry, Josephson junctions, and integrated-optics. We are looking for one additional student in this area.

D. REVERSIBLE PROGRAMMING LANGUAGES

In this area we have made considerable progress, both in the development of abstract mathematical models and in the development of a reversible programming language. The first versions of this language, written in LISP, were successfully run on several test problems; however, they are not yet adequate for use in systematic research. An undergraduate student, Guy Vachon, will be writing a new version of these programs over the summer (he plans to use APL). We expect to generate a LISP version of his programs in the fall. The concept of reversible programming is to realize each computing function as a reversible module that operates on two lists of variables. If a global parameter is set to "forward," then each module will replace the values of the variables in the second list with a given function of the values of the variables in the first list. If the same global parameter is set to "backward," then the module will replace the values of the variables in the first list with the inverse function of the values of the variables in the second list. In the backward mode PROG itself runs backwards, and subroutines proceed from their entry points to their calling sequence, etc. Several such systems have been implemented in LISP, but it is clear that a fair amount of work will be needed in order to come up with a smooth and esthetically pleasing version of a reversible programming system.

E. REVERSIBLE SYSTEMS THEORY

In the field of reversible computing, Tom Toffoli has written a paper on "Bicontinuous extensions of invertible combinatorial functions," and another comprehensive paper on the general subject of "Reversible computing" is nearing completion. Other ongoing research includes reversible cellular automata, causal networks, and synchronization problems.

F. REVERSIBLE DIFFERENCE EQUATIONS

Physical systems are most often represented by systems of differential equations, which may or may not be solvable in closed form. Of course we can also represent systems by means of difference equations, which may be iterated in order to determine the future state of a system from some initial condition. This is usually done with the aid of a computer, because of the vast amounts of calculation involved. It has been generally thought that computer programs that implement such systems are necessarily irreversible, since at many steps the program encounters round-off or truncation errors, and it seemed that information was irretrievably lost each time this happened. Exceptions had been noticed; for example, Prof. Berthold Horn, in a brief paper on the computation of circles observed that a certain program that computed the circle was reversible, following exactly the same trajectory when run backwards, *in spite of the round-off error.* We have since found more general methods for constructing reversible systems of difference equations in correspondence with given systems of differential equations. Some of the results are quite exciting. For instance, while such processes as thermal diffusion--or processes that undergo exponential decay--appear to be examples of systems that *must* be irreversible when programmed on a computer; yet it is easy to model them so that they are representable by a reversible computer program. This way of modeling them is actually more adherent to the underlying physical reality of the processes than the usual models patterned after relaxation methods. Of course reversibility could be "faked" (though even this meets with considerable difficulties) by saving in some pushdown stack all of the data that would otherwise be lost during a computation, but this is a dead-end approach. Our method consists in using a number of "magic tricks" (cf. the work of D. Greenspan on time-discrete mechanics), so as to obtain a perfectly legitimate set of difference equations for the given system. These equations, though exhibiting, say, the desired exponential decay, constitute a model that is exactly reversible even when run on a computer affected by certain approximation errors.

We expect to continue this work next year; in particular, one student has chosen this area as the subject of his senior thesis.

G. DIGITAL MECHANICS

This area of research involves a general attack on the problem of modeling physical phenomena by means of digital processes. It is an attempt to construct for physical processes abstract models that are entirely quantized, so that the models are exactly isomorphic with a computer simulation of the models. We must contrast this with the common task of approximating the characteristics of a physical system by means of a computer program so that we can make approximate simulations. It is our goal to find exact models that behave like physical systems, yet which are entirely discrete. In this area John Featherly completed his senior thesis on "The small-scale structure of space and time," done jointly in the EECS and Physics Departments.

One of the properties of physical systems is that they conserve certain quantities. Within the limits of accuracy of measurement, these quantities seem to be conserved exactly. We have proceeded to find ways of constructing discrete systems of many types that conserve certain quantities exactly. While such conservative discrete systems were first found in the area of logic (Conservative Logic) we have been able to slowly progress toward abstract dynamical systems that are more adherent to physical constraints, such as cellular automata and systems of difference equations.

To sum up, in digital mechanics we are trying to match new models of computation to physical processes in a way similar to the matching of systems such as the calculus to many physical processes.

H. ROBOTICS, BALANCE, AND WALKING

Robert Giansiracusa has substantially completed an elaborate simulation system of the simple walking robot design known as the Hinge System. This system was first implemented on the PDP-10, but was transferred to the LISP Machine recently. It consists of an accurate model of a one-degree-of-freedom robot which can walk (hop) in a two-dimensional world. The model is accurate to the point of taking into account elasticity, friction, deformation of the joint, etc. The robot consists of two rigid limbs, connected by a single joint. The robot sets the torque that is applied to the joint. We have been able to show that such a robot can get up gracefully from a prone position, that it will be able to balance itself for any reasonable angle of the joint, and that it will be able to hop around, all through the use of the one joint, with one degree of freedom. The generalization from two dimensions to three seems to be straightforward. By adding a limited second degree of freedom to the existing joint it will be possible to operate in a normal three dimensional environment. The problems that are elucidated by this project are those connected with the necessity of planning a long sequence of actions in order to achieve a desired goal. In some sense it is the problem of constructing a servomechanism that operates in a multi-dimensional space where only certain trajectories are *reasonable*. It seems that the key to success in this area will be to operate along certain known trajectories in order to follow a familiar path to a desired state. We are confident of being able to both program the simulation system so that it performs satisfactorily, and of being able to build and program a working model, which would weigh about five kilograms.

I. ROBOTICS, LEARNING FROM EXPERIENCE

Underlying our interest in the Hinge system is the question of learning. We will be using the Hinge simulation system to test our concepts of learning from experience. This type of learning is the kind that is involved when an animal learns a physical skill, such as a dog learning to walk on his front paws. Our main approach can be thought of as a memory-intensive solution, in that our learning system will remember a great deal of what it experiences, and it will have a simple method of finding in its memory that experience that is most similar to the one it is looking for. We expect that the size of the active memory that will be used in remembering past experiences will be on the order of 100,000,000 bytes. The general strategy of the system will be to try to get onto a familiar trajectory so that it can predict the consequences of possible actions. We hope to find ways to compactify useful information and discard that which is not really useful, but the main emphasis will be on the use of memory, when memory contains nearly everything from the past. We believe that we should be able to acquire skills at a rate not too different from a human or a rat, so that a simple physical skill should not take much more than tens to hundreds of hours of computer time for it to reach a high state of development. Consider how long it takes one to acquire skill as a gymnast, or how long it takes to perfect a tennis serve.

J. SEMI-INTELLIGENT CONTROL SYSTEMS

We have written a substantial part of a proposal to conduct research on the design and construction of integrated systems consisting of microcomputers combined with machinery. The goal is to demonstrate the feasibility of making machinery more dependable, more energy-efficient, less expensive to produce, easier to maintain and, in general, better human-engineered. The suggested approach is to add semi-intelligent monitoring, control, and communication through the extensive application of VLSI technologies. The resultant integrated systems of microcomputers and machinery would have a great amount of self-awareness and internal adaptability; these features would be used to operate the machinery more efficiently and to protect it from damage, to allow the machinery to cope with component failure and changing conditions, and to assist maintenance personnel in both the care and the repair of the machinery.

The research will involve three major phases: (*a*) the construction of design tools for the rapid development, out of existing one-chip microcomputers, of control systems applied to machinery; (*b*) the design of new VLSI circuitry for such control systems and the actual implementation of a number of example systems; and (*c*) the development of standard guidelines, technologies, and components, in order to provide a solid foundation for the dissemination of this methodology as well as permit a fast, "cookbook" approach to the implementation of a variety of systems.

In conclusion, we anticipate that our research will eventually provide methods for constructing complex systems of great efficiency, robustness, and dependability *in a systematic way*, that is, essentially by interconnecting at a *functional* level a number of *goal-oriented* pieces of hardware, thus making detailed, low-level programming in general unnecessary.

KNOWLEDGE-BASED SYSTEMS GROUP

Academic Staff

W. A. Martin, Group Leader

Research Staff

G. P. Brown
G. Burke
L. B. Hawkinson

V. E. Lewis
G. R. Ruth

Graduate Students

G. Faust

Undergraduate Students

P. Koton

Visitors

E. Ejerhed

Support Staff

M. Briggs

KNOWLEDGE-BASED SYSTEMS GROUP

A. INTRODUCTION

The long term goal of our group is the creation of interactive computer systems which can solve problems of a type currently handled by expert consultants. Our interest is in problems which can be stated and discussed verbally, without the use of pictures or complex special purpose notations or graphics. Examples occur in medicine, law, operations management, and programming.

Two topics are the focus of our current efforts. The major topic is natural language question answering. We also have some work on software construction and maintenance. In addition, our software systems are used by the Medical Decision Making Group and we work closely with them.

B. REPRESENTATION OF KNOWLEDGE

In the spring of 1978 we implemented a prototype question answering system. The issues and level of ambition arising from that experience are discussed in Martin (1979a). We were reinforced in our belief that many questions can be answered in a straightforward manner provided the system knows a large number of domain specific facts. For example, ships carry missiles in two senses, as weapons or as cargo.

The rate at which these facts could be added to the system was a major bottleneck to progress. It is obvious that the fact addition process must be semi-automated in order to reduce the cognitive burden on the system builder. Otherwise, his progress is too slow and he makes too many mistakes. As facts were added to the prototype system, a number of specific issues arose which needed to be resolved in a general way. The implementation provided a basis for rethinking our ideas on knowledge representation.

1. Philosophical Issues in Knowledge Representation

During August 1978 through January 1979, Prof. Martin pursued basic issues which arise in the representation of knowledge in frames or semantic networks. He has produced a draft document which is being used as a working guide for the next implementation.

The proposed formalism is a further development of the general directions suggested in Martin (1979b). Objects may be described from several different viewpoints, perhaps at different levels of abstraction. A "frame" is used to describe a particular viewpoint of an object. Some frames are generic, others describe individuals.

A unique and far reaching feature of the system is that every frame which describes an individual must be a role in some other frame. This means that every concept of an individual arises from the structural decomposition of some other individual--we make the exception of the real world. The question of existence of abstract individuals, i.e. places, is reduced to a discussion of the validity of a certain structural decomposition of the world.

This leads to an implemented language which replaces "contexts" with explicit structural descriptions. Objects don't exist in abstract spaces or contexts, but rather as part of the structure of other objects.

Another consequence of this view is that an ontology can be developed, based on whether an object, A, can be identified only by identifying the individual description of which A's description is a role or an individual whose description is a role in A's description. This leads to the basic categories:

a) attribute, i.e., length--can only be identified as a role of an individual

b) state-process, i.e., run--can only be identified by knowing what individuals fill its roles

c) relational-characterization, i.e., further--can only be identified from this description as a role of an individual, but has other structural description, i.e., man, continuing enough information for identification

d) others

A programming system, OWL II, is being implemented to serve as a basis for these ideas. The following describes the lowest level components of this system.

C. XLMS

XLMS is a new implementation of Hawkinson's Linguistic Memory System, one of the fundamental components of OWL. XLMS is the product of a major redesign/reimplementation effort intended (1) to bring LMS into line with current notation, conventions for low-level knowledge representation, and models of linguistic memory; (2) to improve the structure and organization of the code and documentation; (3) to eliminate minor anomalies; and (4) to produce a user's manual that can be readily kept complete and up-to-date.

XLMS is built directly on LISP, and may be thought of as extending LISP's repertoire of object types to include object types designed for natural-language-based knowledge representation. All such object types are specialized types of node. The "node" is the basic building-block in XLMS; it plays somewhat the same role in XLMS as do list cells and atoms together in LISP. Every node has a unique name, and it may also have any number of attachments. The name of a node not only identifies it, but typically provides much (if not all) of its meaning, determines its position in organized collections of nodes, and often is processed as a formal expression; the name of a node is the node's essence, and the significance of this name strongly distinguishes the XLMS node from ordinary nameless semantic network nodes. The name of a node has two parts: the "genus", itself a node, and the "specializer", a node or an atomic symbol. The genus and specializer of a node are almost always semantically meaningful, though their meaning is almost never supplied by XLMS, but rather by some system built on top of XLMS. The genus of a node almost always plays a crucial role in its interpretation; what a node represents is usually a special case of what its genus represents. A node with an atomic symbol specializer usually represents a particular meaning of that symbol in some language. A node whose specializer represents an entity or category usually represents an aspect of that entity or category, or a related entity or category. Any type of object may be attached to a node, provided that an appropriate attachment relation is used. An "attachment relation" may be thought of as a label on a directed arc from a node to some attached object. One or more objects attached to a node x by a particular attachment relation r constitute a "zone", the r zone of x.

There are two important relations defined on nodes that provide a basis for organizing sets of nodes, in particular, all the nodes in an XLMS memory. The "under" relation on nodes may be defined roughly as follows: a node x (other than SUMMUM-GENUS) is under (a) its genus, (b) anything its genus is under, and (c) any node y with the same genus as x and such that the specializer of x is under the specializer of y. Case (c) is called "derivative subclassification." The "before" ("canonical order") relation on nodes is a particular total ordering, defined in terms of genuses and specializers and strongly related to the under relation. When x is under y, y is said to be a "superior" of x. If a node x has one or more superiors in a set S, there is always a least such superior in S. The set of all node-in-S/least-superior-in-S relationships determines the "superior structure" of S; it is a forest (i.e., a set of trees). If a set S' is generated by using every element of S as the specializer of a node whose genus is some fixed node g, then the superior structure of S' will, because of derivative subclassification, be isomorphic to the superior structure of S. In an LMS memory, there is a node SUMMUM-GENUS which is a superior of every other node. Thus the superior structure of the nodes in an LMS memory is a tree with root SUMMUM-GENUS. This "node tree" constitutes the primary organization of an LMS memory.

XLMS provides three specialized types of node for general (public) use: the concept, the finished sequence, and the LMS number. XLMS supports these types as if they were basic LISP object types, that is, XLMS can read and print them in special notations, can deal with occurrences of them as constants in code and as elements of S-expressions, and provides appropriate operators to manipulate them and iterators to iterate through sequences associated with them. A "concept" is conceptually similar to a node, but has a name with three parts: an ilk, a tie, and a cue. An ilk is analogous to a genus, and a cue to a specializer. A tie is a node that expresses the relationship between the meaning of a concept and the meaning of the parts of its name. (Martin has identified a small set of primitive ties for use in OWL, which are predefined in XLMS.) A "finished sequence" is a sequence of nodes and atomic symbols that is represented as a node whose name is derived from an initial subsequence of elements of a length sufficient to make that name unique in a particular state of an XLMS memory. An LMS number is a particular representation of a number as a concept, the name for which is structured, at present, in accordance with the common English expression for that number.

The original goal of the XLMS effort ("X" stands for experimental) was to build a pared-down and well-documented version of LMS suitable for use by students in Martin's natural language and knowledge representation course. But it soon became clear that what was really needed was a generally improved version of LMS, not a pared-down one; thus a major reimplementation was undertaken. Except for a few loose ends, the XLMS implementation is finished. Instrumental in the rapid and smooth implementation of XLMS were: (1) the insights gained from four years' experience with prior versions of LMS; (2) the completion of a draft users' manual prior to any coding; (3) the design effort expended to make XLMS as operationally uniform as possible; (4) full use of LSB (see the next section) to achieve a high degree of program modularity; (5) adherence to the precepts of structured programming (which was much facilitated by Szolovits' FOR package); and (6) the use of data abstraction techniques, in the spirit of CLU. XLMS is currently being used by several members of the Knowledge-Based Systems and Medical Decision-Making groups.

D. LSB

Despite the fact that LISP has been the implementation language of choice for many large and complex programs, LISP systems generally provide little support for strong modularization of programs. Inevitably, most large LISP programs have not been well modularized, and have tended to become difficult to modify and to move to new environments. Many large-LISP-program builders, notably Winograd's response to this problem, were to propose the use of a "programmer's apprentice' to aid in getting past these barriers; indeed, there have been several ambitious efforts aimed at building such apprentices. A more staightforward response to the problem is to push back the barriers by implementing large programs more modularly. CLU is a complete programming system that supports a very high degree of modularization of all aspects of LISP programs, including documentation.

The "degree of modularization" of a program is an informal measure reflecting (a) how little repetition of detail there is in the source text for the program, especially non-localized repetition, and (b) how consistently closely related components at a similar level of abstraction have been grouped together. From this definition, it follows that a programming system supports a high degree of modularization if it provides facilities for deriving every non-source-text manifestation of a program (or part of a program) from the source text. Thus, for example, documentation at all levels, environment-specific versions, "interface descriptions," and declarations needed for compilation should all be derivable from a single source text. (Of course, to enable documentation and environment-specific versions to be derivable from source text, source text must include pieces of documentation and environment-specific code. Note, however, that pieces of documentation need not be components of definitions, as they normally would be in INTERLISP, LISP Machine, and EMACS programs.) Source text should be organized so as to make for good derived documentation, with due regard for continuity and completeness; code in source text should be considered subordinate to documentation.

A LISP program that takes maximal advantage of LSB to attain a high degree of modularization has the following structure. A "program" is a particular collection of separate but cooperating packages. A "package" consists of a set of modules plus a package description. A "package description" is a small piece of source text that specifies, among other things, (a) the "package name" (an atomic symbol other than PACKAGE), (b) the location of each of the modules in the package, and (c) the packages, if any, upon which this package is "built." The source text for a "module" is divided into "sections," each of which contains pieces of documentation interspersed with LISP forms, especially definitions. (A section of a module might contain, for example, a set of definitions constituting a CLU-like data abstraction.)

A "definition" is an operation that creates (or redefines) a "defined object," a permanently named object such as an ordinary operator, a special variable, a defined constant (i.e., T or NIL), or a type. LSB provides definition facilities which are substantially more elaborate, more general, and more consistent in format than those typically available with LISP systems. A special variable may be typed and may be given a default initialization. An operator may be defined as a routine, an open-codable routine, or a macro. (An "open-codable" routine is a routine, calls to which may be compiled "open.") Operator argument variables may be typed; may be declared UNQUOTED (implying that corresponding arguments are not to be automatically interpreted); may be declared OPTIONAL, with or without specification of a default argument form; and may be made to correspond to other than precisely one actual argument. Special facilities are provided for binding and initializing non-argument variables and for making certain auxiliary declarations in operator definitions.

LSB offers a choice of scope for defined objects, all static in nature, but none as rigid as lexical scoping. As part of each definition, the "scope class" of the defined object is specified, determining where its given name and details of use should be known. A PRIVATE object is known only within the module where it is defined; a PACKAGE object is known throughout the package in which it is defined; and a PUBLIC object is known not only throughout its defining package, but also throughout all packages built on this package. (Note that the "built on" relation is not transitive.) Every package is implicitly built on the GLOBAL package, so that PUBLIC objects in the GLOBAL package are known everywhere. In addition to the scope classes for defined objects, there is also LEXICAL, which applies only to locally bound variables.

When the source text for an LSB module is compiled by a LISP compiler, many distinct partial (and often environment-specific) manifestations may be derived: a compiled version (the sequence of top-level LISP forms in compiled form), public documentation, package ("internal") documentation, public declarations (details of use for PUBLIC objects), intrapackage declarations (details of use for PACKAGE objects), "bootstrap" code, etc. Any such derived manifestation passes through a "diversion stream," and may then be either put out as a file or further processed in some way. Documentation and manifestation-specific code typically appear in source text in the form of curly-bracket-delimited "conditional inclusions" specifying one or more diversion streams. Pieces of environment-specific code appear as conditional inclusions with environment-specifying "inclusion tests." LSB should, but does not yet, provide one or more programs to aggregate particular kinds of module-specific manifestations into: runnable programs, program or system manuals, package descriptions with selected amounts of detail, public and intrapackage declarations or whole packages, the declarations needed for the compilation of a particular module, ready-to-run programs, listings, program analyses like the INTERLISP Masterscope program, etc. LSB should also be capable of keeping such aggregations up-to-date.

Lowell Hawkinson was the original designer of LSB and, with John Thompson, implemented early versions of its basic components. Glen Burke has since refined the design substantially and has implemented LSB for general use as an extension of MACLISP; Hawkinson and Burke are now designing the not-yet-implemented components of LSB and are putting together a user's manual. Burke is planning to add code to LSB for the MIT LISP Machine and for the NIL dialect of LISP.

E. DATABRIDGE: A SYSTEM FOR QUERIES ABOUT DATA BASES

In order to test our knowledge representation language and question answering system we have been building a system for answering natural lanuage queries about the structure and use of data bases. This system is called Databridge, because it is intended to help bridge the gap between the user and the world of formal data bases. Work on the Databridge project started January 1977, and an initial version was operational in July 1977. In the past year, work has focused on the Model Acquisition Module, which is reported on here.

The goal of the Databridge system as a whole is to provide a facility to answer typewritten English questions about one or more data bases. This would permit a user to determine which of a given set of data bases met his or her needs. Note that the queries we wish to handle are not about specific entries in a data base, but about the *type of contents* that the data base contains. Thus, a sample question would *not* be (1) below; instead, a typical question might be (2).

1. Which ships were decommissioned in 1975?

2. Do any of your data bases have information on merchant shipping?

We approach natural language queries with the conviction that a natural language query system needs rich semantic support. To this end, the Databridge question answering facility will draw on extensive descriptions of the subject data bases. The representation formalism used is OWL-II; a sample description has been developed for the Worldwide Analytic Research Project (WARP) data base developed by Air Force Data Services Center. (Worldwide Analytic Research Project (WARP), Data Base Documentation and User's Guide, Office of the Secretary of Defense, Office of the Director, Planning and Evaluation, Washington, D.C., January 1977.) WARP contains demographic, political, socio-economic, military, and commercial information of about 195 countries.

When one thinks about the actual construction and use of a data base *about* data bases, a question that comes immediately to mind is what happens if the data in any of the subject data bases should change. Perhaps a data base is added or information about a known data base must be updated. It would be desirable to have update of the knowledge base occur as easily as possible and to require as little as possible in the way of specialized knowledge representation skills. In short, we wish to take the OWL knowledge base designer out of the design and update phases of Databridge knowledge base development.

To meet this requirement, we are designing and implementing a Model Acquisition Module for Databridge. Information about a data base is acquired *from the data base designer*. This is done interactively in a combination of menu and short-answer frames. The model acquisition interview is divided into three phases, which we describe briefly in turn.

In Phase 1 of Model Acquisition, a model of the structure of the data base is built and represented in OWL-II. In July 1978, work began to convert the existing implementation of Phase 1 to use the SDM data model. SDM (Semantic Data Model) was developed by Prof. Michael Hammer and Dennis McLeod. (See Hammer, M. and McLeod, D. "The semantic data model: A modelling mechanism for data base applications," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Austin, Tx., 1978, and McLeod, D. "A Semantic data base model and its associated structured user interface," MIT/LCS/TR-214, MIT, Laboratory for Computer Science, Cambridge, Ma., August 1978.) We have found SDM attractive because it supplies a highly differentiated model that has intuitive appeal.

The SDM model acquired in Phase 1 forms a basis for acquisition in subsequent phases. In Phase 2, the modeller is asked to enter instances of what we are calling *major classes*. Major classes are classes of data base entities around which a user community structures its information. ("Users" here are prospective users of the individual data bases.) For example, in the WARP data base a major class is the set of countries. In interpreting queries, it is useful to know not only that WARP has country information but also that it has information on the U.S., U.S.S.R., Japan, etc. This allows straightforward answer-finding for queries such as (3).

3. Do you have anything on Iraq?

Finally, we come to phase 3 of the Model Acquisition interview. Phase 3 acquires information relevant to the *use* of a data base. Examples of what we are calling *usage attributes* are the sources of a piece of data, the time period for which it is relevant, the names of individuals responsible for its maintenance, and its security classification. In all, 18 different usage attributes may be set for the data base as a whole, for particular SDM classes and attributes that were defined in Phase 1, or for additional classes that may be defined as part of Phase 3. The usage model acquired in Phase 3 will permit a user of Databridge to find out not only where his or her information may be located, but also how to go about arranging access to the information.

This completes our survey of the three parts of the Databridge Model Acquisition interview. Gretchen Brown is working on the Model Acquisition implementation. Implementation of Phase 1 is now complete, and substantial progress has been made on the implementation of Phase 2 and 3. In addition, a change facility has been partially completed. The goal is to give the modeller a way to change anything that may have been entered in the course of the console session, as well as to update entries in the completed data base model. We expect to see this round of implementation finished in early Fall 1979.

F. DIALOGUE MODELLING

The dialogue modelling project was begun as part of the work on the OWL-I knowledge representation language; active development of the dialogue model was completed in January 1977. A number of issues were explored at the discourse level of natural language, i.e., at the level of the connections between utterances. More specifically, we were interested in modelling the type of dialogue that would occur between a user and an automatic programming system. An early report on the model and implementation was given in MIT/LCS/ TR-182, (Brown, Gretchen P. "A framework for processing dialogue," June 1977).

In the past year, work in this area has continued on a reduced scale, with the intent of winding up the projects. A paper by Gretchen Brown on the general philosophy of the dialogue model has been accepted for publication. (Brown, Gretchen P., Linguistic and situational context in a model of task-oriented dialogue, to appear in L. Vanya and K.J.J. Hintikka (eds.): *Models of Dialogue*, D. Reidel, Dortrecht, Holland, to appear.) In addition, a taxonomy of indirect speech acts based on the dialogue modelling work is reported in a paper that has been submitted for publication.

G. AUTOMATIC PROGRAMMING

The ultimate aim of Automatic Programming is to make the computer directly available to the end user with a problem to solve. The development of a non-trivial data processing system may be modelled by the following sequence of steps:

Step 1: The end user explains his desires to a consultant.

Step 2: The consultant produces a data processing requirements specification.

Step 3: A software system analyst designs data structures and a framework of processing steps to be used to accomplish the required tasks.

Step 4: A programmer writes code to implement the design.

Step 5: A compiler processes the resultant programs, producing machine level code.

Traditionally, all steps save the last are performed manually (by humans); communication between neighboring steps is imperfect. This means that the software development process is slow and error-prone. The end user is far removed from the final result and the programmer, who writes the code in Step 4, is not generally cognizant of the real problem being addressed. Clearly, the more steps that are automated, the more successful will be the match between what the end user wants and what he gets.

1. Scope of ProtoSystem I

The ProtoSystem I project is concerned with the automation of Steps 3 and 4. Our goal has been to provide entry into the software development process at Step 3. Given the above model of this process the ProtoSystem I project has evolved into the study of a Very High Level Language, HIBOL (High Level Business Oriented Language, also known as SSL (System Specification Language), and an underlying automatic design and implementation system for it.

In this approach to automatic programming the user describes his application in the VHLL (Very High Level Language) and the automatic programming system determines the designs of data objects and processing and then generates implementing code (in some "high-level" language, such as PL/I) automatically. The VHLL treats a particular data processing domain, capturing and simplifying its essential aspects so that common processing requirements are easy to describe. In order to allow full flexibility in the automated design and implementation process, the language should not permit too detailed a specification of data objects or processing; it should permit only functional specification of the application.

The HIBOL language supports the description of applications in the business data processing domain. It is non-procedural, "goto-less" and is not universal. It provides a stylized way of specifying file-oriented batch-processing systems.

2. Status of ProtoSytem I

A preliminary version of the Protosystem I automatic programming system (implemented in the LISP language) is now running on the Mathlab PDP-10 computer. It can ingest a HIBOL description of a file-oriented batch data processing system, and without human intervention, design, implement and generate PL/I and OS/260 JCL code for a collection of programs comprising the described application.

ProtoSystem I consists of four major modules. The Parser accepts HIBOL descriptions and translates them into a machinable form, producing "first cut" computations and "first cut" data sets. The Structural Analyzer (a utility module) models the properties of these primitive entities. The Optimizing Designer arranges and combines computations into programs and data sets into multi-field data sets. It also determines data set organization, blocking factor, key sort order and access technique.

The Optimizing Designer endeavors to design these data objects and programs in such a way as to minimize overall run-time cost. It makes continual use of the Structural Analyzer to model the properties of tentative computation and data set designs. (The greater part of our research effort over the last few years has been addressed to the optimization of data processing system designs.) When the Optimizing Designer is done, the finished design is passed to a PL/I Code and JCL Statement Generator.

The theory and technology involved in this project have been written up in an extended monograph (book) entitled "A Very High Level Language for Business Data Processing."

3. Very High Level Language Research

One of the most important components of our recent research in automatic programming has been the development of the HIBOL VHLL. Originally, the scope of HIBOL was purposely restricted to the simple (but important) class of applications involving only batch processing of large uniform files of data, so that development of a supporting automatic programming system would be practically feasible. With that development completed, it was deemed worthwhile to extend the semantics of HIBOL to cover a much broader range of applications in order to more completely serve the needs of real-world large system developers.

The major thrust of this HIBOL enhancement work is the development of a system specification language that will support the description of on-line and transaction-driven data processing systems, data base management and audit trail, checkpoint/restart and other data integrity/security requirements. The emphasis here is on providing a rich and concise requirements definition/system specification medium of near-term utility. The development of supporting system maintenance, enhancement and (eventually) automatic programming tools is a longer-term consideration.

4. Unprogramming

In parallel with our HIBOL enhancement effort, research is being conducted in the area of "unprogramming" existing high-level language implementations. The prime candidates for the application of the advanced specification technology described above are predominantly large existing systems in need of revision or revamping. In order to receive the full benefit of our new specification technology system would have to be re-described in HIBOL in its entirety. A manual approach would be time-consuming, expensive and error-prone. However, making use of the recent work of Waters, we feel that a semi-automated system that would extract the plans from the existing code and, in turn, translate these plans up to HIBOL could be developed. We are actively engaged in the development of such a system.

Publications

1. Brown, Gretchen P. "Linguistic and situation context in a model of task-oriented dialogue," in Models of Dialogue, Vaina, L. and Hinitikka, K.J.J. (Ed.), D. Reidel, Dortrecht, Holland.

2. Martin, W.A. "Some comments on EQS, a near term natural language data base query system," Proceedings of the 1978 ACM Annual Conference, Washington, D.C., December 4-6, 1978, 156-164.

3. Martin, W.A. "Descriptions and the specialization of concepts," in Artificial Intelligence: An MIT Perspective, Winston and Brown (Eds.), MIT Press, Cambridge, Ma., 1979.

4. Martin, W.A. "Discussion of 'Review of Natural Language Processing'," in Research Directions in Software Technology, Wegner, Peter (Ed.), MIT Press, Cambridge, Ma., January 1979, 842-844.

5. Ruth, Gregory R. and Hammer, Michael, "Automating the software development process," in Research Directions in Software Technology, Wegner, Peter (Ed.), MIT Press, Cambridge, Ma., January 1979, 767-790.

Talks

1. Martin, W.A. "Natural language data base query,"
   panel discussion at the 1978 SIGMOD Conference, Austin, Texas, July 1978;
   IBM San Jose Research Laboratory, San Jose, California, August 1978.

2. Martin, W.A. "Applications program research at MIT," Teknikron, Oakland, California, August 1978.

3. Martin W.A. "Natural language research at MIT," Stanford Research Institute, Menlo Park, California, August 1978.

4. Martin, W.A. "Applications program research at MIT Laboratory for Computer Science," Raytheon Submarine Signal Division, Portsmith, Rhode Island, March 1979.

Theses Completed

1. Levin, Beth, "Instrumental WITH and the control relations in English," S.M. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., February 1979.

Theses in Progress

1. Koton, P.A. "Simulating a semantic network in LMS," S.B. thesis, MIT, Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., expected January 1980.

PROGRAMMER'S APPRENTICE GROUP

Academic Staff

Howard E. Shrobe	Richard C. Waters

Graduate Student

Charles Rich

Undergraduate Student

David Chapman

PROGRAMMER'S APPRENTICE GROUP

Howard Shrobe finished his Ph.D. thesis on reasoning about programs in August 1978. He has redesigned and implemented an improved version of his reasoning system on the MIT LISP machine. The new reasoning system incorporates several new insights which have been developed over the past year. It employs a discipline of explicit representation of its control state. This facilitates the creation of flexible control strategies which can use information in the deductive context in order to help decide what to do next. In addition, the system employs the Truth Maintenance System (TMS) developed by Jon Doyle to maintain a record of the logical dependencies between segments of the program. The TMS is responsible for maintaining consistency in the information describing the program. For example, if assumptions are withdraw the TMS will automatically withdraw all the information which depended on the withdrawn assumptions. Similarly, when the description of part of the program is changed the TMS will remove all the information which no longer has logical support. These features provide a starting point for interactive support of evolutionary programming.

Richard Waters finished his Ph.D. thesis on program analysis in August 1978. Since then, he has completed the implementation of the program analysis system described in the thesis. This system consists of two main parts, a surface analyzer which converts FORTRAN programs into a language independent internal form called a "surface plan", and a deep analyzer which analyzes the surface plan in terms of a set of Plan Building Methods (PBMs). One of the most interesting aspects of the analysis method is the way it handles loops. It analyzes a loop as built up out of a "composition" of stereotyped looping fragments. The central idea is that, logically, these fragments are in fact simply composed together because the only interaction between two fragments is that one creates data used by the other. The idea of "temporal abstraction" has been developed by Waters and Shrobe in order to make the notion of composition of loop fragments precise. The process of temporal abstraction views the sequence of values transmitted between two parts of a loop as a single data object called a temporal sequence. The loop fragments themselves are then viewed as logically equivalent to non-looping segments interacting by means of these temporal sequences. The deep analyzer is programming language independent. In conjunction with a surface analyzer written by Charles Rich, it can be used to analyze LISP programs.

Charles Rich is finishing up his Ph.D. thesis on representing and cataloging knowledge about basic algorithms and data structures. He expects to be done in August 1979. His thesis focuses on two typical implementations of an associative retrieval data base. The first example program is a simple hash table which uses an array of Lisp-style association lists. The second example program stands at the limits of his current cataloging effort. It is a complicated data base implementation (similar to the one used in early AI programs, such as CONNIVER) which uses ordered lists and a multiple intersection strategy. These programs are analyzed with a hierarchy of descriptions which make explicit the way in which standard programming ideas are applied in these particular programs. The standard programming ideas are formalized as "plans." For example, the Aggregation plan captures the common idea underlying summation, set aggregation, and accumulating objects in a list. An example of a less abstract idea which is formalized as a plan is the use of "flags" in programming. Flags are used to encode information about splitting in control flow which can be recovered later by testing the flag and splitting again. The result of Rich's thesis will be a library of programming plans which has applications in the programmer's apprentice for the automated analysis, synthesis and verification of programs.

Shrobe is working on a demonstration system which will interact with his reasoning system. In this system, the user interacts with the reasoning system directly in terms of plans. These plans are graphically displayed, and can be modified by the user. The graphical representation is in many ways much more convenient than program text. For example, it eliminates many of the problems associated with variable scoping and binding. It also corresponds closely to the internal representation maintained by the reasoning system. This system will be a prototype of a development environment in which the user displays and edits his code graphically while a reasoning system analyzes it logically.

Waters is also working on a demonstration system which will exhibit some of the capabilities of the proposed apprentice system. This demonstration system is essentially an intelligent semantically based editing system. The basic idea behind this editor is that the user will be able to modify his program in two ways. First, he can edit it by adding and deleting characters as in an ordinary text editor. Second, a program can be modified by issuing semantic requests based on the PBMs (for example a user might say: "add this filter into that loop" or "change the initialization of that accumulation to ..."). In order to support these two modes of interaction, the editor will maintain two representations of each program being edited. It will have a representation of the program as text. In addition, it will have an analyzed plan for the program. The semantic editing requests are based on referring to this plan. If the user changes the text, the program must be reanalyzed in order to determine what the new plan should be. If the user specifies a change to be made to the plan, the editor must determine what changes this will cause in the text. What the user sees when he is working on the program is the program text. The analyzed plan exists in the background in the editor, and in the users mind. The fact that the PBMs correspond to a natural way to think about programs is an important prerequisite of this approach.

As mentioned above, the analysis system needed for this editor is already running. The new module which has to be produced, is a coder which can take an analyzed plan and produce code corresponding to it. This will be used to determine how the program text should be changed when the user specifies a change in the plan. The basic algorithms to be used in the coder are described in Waters' thesis, however nothing has been implement yet. The coder should be constructed this summer.

Publications

1. Rich, C. and Shrobe, Howard E. "Initial report on a Lisp programmer's apprentice," IEEE Transactions on Software Engineering SE-4, 6 (November 1978).

2. Rich, C., Shrobe, Howard E., and Waters, Richard C. "Computer aided evolutionary design for software engineering," MIT/AIM-506, MIT, Artificial Intelligence Laboratory, Cambridge, Ma., January 1979.

3. Shrobe, Howard E. "Reasoning and logic for complex program understanding," MIT/AI/TR-503, Ph.D. dissertation, MIT, Artificial Intelligence Laboratory, Cambridge, Ma., June 1979.

4. Shrobe, Howard E. "Explicit control of reasoning in the programmer's apprentice," Proceedings of the Fourth International Workshop on Automated Deduction, University of Texas at Austin, February 1979.

5. Shrobe, Howard E., Waters, Richard C., and Sussman, Gerald J. "A hypothetical monolog illustrating the knowledge underlying program analysis," MIT/AIM-507, MIT, Artificial Intelligence Laboratory, Cambridge, Ma., January 1979.

6. Waters, Richard C. "Automatic analysis of the logical structure of programs," MIT/AI/TR-492, Ph.D. dissertation, MIT, Artificial Intelligence Laboratory, Cambridge, Ma., December 1978.

7. Waters, Richard C. "A method for analyzing loop programs," IEEE Transactions on Software Engineering SE-5, 3 (May 1979), 237-247.

Accepted for Publication

1. Rich, C., Shrobe, Howard E., and Waters, Richard C. "An overview of the programmer's apprentice," Proceedings IJCAI-79, Tokyo, August 1979.

2. Shrobe, Howard E. "Dependency directed reasoning in the analysis of programs which modify complex data structures," Proceedings IJCAI-79, Tokyo, August 1979.

3. Waters, Richard C. "A method for automatically analyzing programs," Proceedings IJCAI-79, Tokyo, August 1979.

Theses Completed

1. Shrobe, Howard E. "Reasoning and logic for complex program understanding," Ph.D. dissertation, MIT, Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., August 1978.

2. Waters, Richard C. "A method for automatically analyzing the logical structure of programs," Ph.D. dissertation, MIT, Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., August 1978.

Theses in Progress

1. Rich, C. "Representation for programming knowledge and applications to recognition, generation, and cataloging of programs," Ph.D. dissertation, MIT, Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., expected June 1980.

Talks

1. Rich, C. "Using overlays to represent programming knowledge," Program Transformation Workshop, USC/ISI, Marina del Rey, Ca., January 8-10, 1979.

2. Waters, Richard C. "Plans and plan building methods in the programmer's apprentice," Brown Univ., Providence, RI, April 16, 1979.

3. Shrobe, Howard E. "Dependency directed reasoning for computer aided evolutionary design," IBM Thomas J. Watson Research Center, Yorktown Heights, NY, June 11, 1979.

PROGRAMMING METHODOLOGY GROUP

Academic Staff

B. H. Liskov, Group Leader
I. G. Greif

Research Staff

R. W. Scheifler
P. Johnson

Graduate Students

R. R. Atkinson
V. A. Berzins
T. Bloom
W. C. Gramlich
M. P. Herlihy
T. O. Humphries
D. Kapur
J. E. Moss
J. C. Schaffert
C. R. Seaquist
L. A. Snyder
M. K. Srivas
E. W. Stark
J. M. Wing

Undergraduate Students

M. D. Allen
L. R. Dennison
R. M. Knopf
P. J. Leach
B. J. Mirrer
C. L. Mullendore
J. L. Zachary

Support Staff

S. Barefoot
J. Jones
A. Rubin

Visitors

A. Merey
U. Montanari
J. Peterson

A. INTRODUCTION

Our major research effort this year has been in the area of distributed systems. We have focused on identifying linguistic primitives that would support the programming of distributed programs. We have also investigated the design of a new CLU compiler and runtime system, with the goals of efficiency and transportability. The primitives for distributed programs will be made available as extensions to CLU, and the extended language is intended to run on the nodes of a network of computers; the new CLU implementation supports this effort. Much of this work is being done jointly with the Computer Systems Research Group.

In conjunction with the Computer Systems Research Group, we sponsored an invitation-only workshop on Distributed Systems. This workshop was held at the Harvard University Faculty Club on October 12 and 13, 1978. Approximately 25 leading workers in the field assembled for two days to discuss research topics and direction. A report on the workshop has been published in Operating Systems Review [16].

We have continued our work on the current CLU system. A reference manual for CLU is now available, and the implementation has been moved to the PDP/20. In addition to this work on CLU, we have completed a study of synchronization primitives, the design of a new machine architecture to support object oriented languages, and the definition of a specification language that permits specification of mutable data abstractions.

In the next section we discuss the status of the present CLU implementation. Section C describes our work on primitives for distributed computing, while Section D discusses the new CLU implementation. The final three sections describe our work on synchronization primitives, object oriented machine architecture, and specification techniques.

B. CURRENT CLU IMPLEMENTATION

During the past year substantial work has been done on the current CLU system. A real-time, display-oriented text editor, with a number of CLU-related features, has been written in CLU. Two improvements to the compiler have resulted in a reduction of the average module compilation time by a factor of four. Major portions of the CLU library, as well as a number of related utility programs, have been implemented. With the design of CLU essentially complete, a preliminary version of the reference manual has been published [14] and circulated for comments. Since its publication, several new data types and an own variable mechanism have been added, and we are in the process of revising the manual for final publication. In addition, the CLU system has been sent to a number of other research groups, both in the United States and Europe.

The first improvement to the compiler was a rewrite of the code generator. The previous code generator transformed the syntax tree for a module (as constructed by the parser and modified by the type-checker) into textual output in a macro language called CLUMAC, and ran a CLUMAC assembler as an inferior process to generate the actual object code. The new code generator produces object code directly from the syntax tree. This eliminates the costly intermediate translation to CLUMAC, and in practice reduces overall compilation time for a module by a factor of two.

The second improvement to the compiler was to augment the code generator to perform, as an option, inline substitution [18] (or open coding) of certain operations of the basic types (e.g., integers, records, and arrays). The only operations chosen were those that, when expanded inline, would result in at most a small increase in code size. The new code generator has been used to recompile the compiler itself, with the result that the new compiler runs twice as fast and is marginally smaller in size. Similar results have been found for other programs. An option was added to the compiler to restrict inline substitution to just those operations that would not increase code size, but in practice this option is not useful, in that simply performing all substitutions generally results in an overall decrease in code size.

In addition to work on the compiler, considerable progress has been made in implementing the CLU library [14]. The library is the repository for information about abstractions and their implementations. For each abstraction there is a description unit (DU) containing all system-maintained information about the abstraction, such as its interface specification. The DU also contains zero or more modules that implement the abstraction, and may additionally contain formal specifications, module interconnection information, and various documentation files. Each implementation contains source and object code, the compilation environment (CE) used to resolve external references in the source code, and other information.

CLU programs tend to be composed of many small modules. Thus the library will contain a great many DUs, and each DU, along with its associated implementations, will contain a number of relatively small pieces of information. Attempting to store this information in a conventional file system could be extremely inefficient in space. For example, one would probably choose to split the information in a DU into several small files: interface specification file, system information file, implementation files, and documentation files. Similarly, one would probably split an implementation into a number of files: source file, object file, system information file, and documentation files. Since files on most systems are composed of an integral number of fixed-size pages, where the page size is fairly large (e.g., 512 or 1024 words), a substantial amount of space can be lost due to breakage. Furthermore, such a library design would not be readily transportable, due to the wide variation in file naming schemes and protection mechanisms.

We have therefore designed and implemented a complete file system for storing large numbers of small files and directories efficiently. About half of this implementation is in assembly language, the rest in CLU. The file system is a tree structure, with the internal nodes being directories and directory-like objects (DUs, implementations, CEs), and the leaf nodes being files. Directory-like objects are implemented through abstract data types, with directories as the concrete representation. Each file is typed with one of five file types (e.g., text or object code). Each entry in a directory has a four part name, consisting of a two part string name of no more than 127 characters, a version number, and a generation (or edit) number. The protection mechanism employed is a variation of access control lists [17].

The CLU file system is contained in a single file of the host file system. This file is divided into logical pages, each page consisting of 1024 words. Each logical page is used to allocate blocks of some fixed size. The block sizes are powers of two, ranging from 8 to 1024 words. Each file (and directory) is composed of zero or more blocks, which need not all be the same size. If a file consists of more than one block, then the file has a header block containing a list of references to all subsidiary blocks. For each block size there is a global free-list chaining together all free blocks of that size. When no more free blocks of a particular size exist, a 1024-word block is broken up into blocks of the needed size.

A directory refers to a file (or subdirectory) with both a unique-id and a slot number. The slot number is used to index into a linear table (allocated from 1024-word blocks) to obtain a reference to the first block of the file, the number of data words in the block, and a flag indicating whether the block is the header of a multi-block file. The unique-id is redundant; it is stored as the first word of every block in the file, to aid recovery in the event of disastrous system crashes.

A command interpreter called SHELL has been written (in CLU) for interacting with the new file system. Many of the commands resemble those of DEC's TOPS-20 EXEC. The commands deal only with the file system; there are no facilities yet for running CLU programs directly from SHELL. In addition, a salvager and garbage collector have been written for use in case of system crashes, as well as an incremental dumper for backing up the file system onto magnetic tape. The CLU I/O facilities have been augmented to deal with this new file system, and hence the CLU-based text editor and the compiler can now access files there.

At present the file system contains only directories and files; DUs and related abstractions are still being designed. Although the file system is still only being used experimentally, the system appears to be nearly as fast as the host file system for most purposes.

C. EXTENDED CLU

Distributed programs that run on nodes of a network are now technologically feasible, and are well-suited to the needs of organizations. However, our knowledge about how to construct such programs is limited. The distributed systems project has undertaken the study of the construction of distributed programs. This study involves

1. Identification of linguistic features that support distributed programming.

2. Experiments in constructing distributed programs.

The linguistic features will be used in the experiments, where they will reduce the effort needed to carry out the experiments. The experiments will provide an evaluation of the features; this evaluation will lead to refinement of existing features, and identification of new ones where appropriate.

In our research, we are influenced by some assumptions about hardware, and about the way in which that hardware will be used. We assume that distributed programs run on a collection of computers, called *nodes*, that are connected by means of a communications network. Each node consists of one or more processors, and one or more levels of memory. The nodes are heterogeneous, e.g., they may contain different processors, come in different sizes and provide different capabilities, and be connected to different external devices. The nodes can communicate only via the network; there is no (other) shared memory. This assumption is in contrast to multiprocessor systems such as CM* [8].

We make no assumptions about the network itself other than that it supports communication between any pair of nodes. For example, the network may be longhaul or shorthaul, or some combination with gateways in between; these details are invisible at the programmer level.

We assume that each node has an owner with considerable authority in determining what that node does. For example, the owner may control what programs can run on that node. Furthermore, if the node provides a service to programs running on other nodes, that service may be available only at certain times (e.g., when the node is not busy running internal programs) and only to certain users. We refer to such nodes as *autonomous*.

The principal consequence of the assumption of autonomy is that the programmer, not the system, must control where programs and data reside. The system may not breach the autonomy of a node by moving processing to it for purposes of load sharing. This attitude distinguishes our approach from others, such as the Actor system [9], where the mapping of a program to physical locations is entirely under system control. Work in the same general area includes [4] and [7], although autonomy is not explicitly addressed.

Our approach to the study of linguistic features is to extend an existing sequential language with primitives to support distributed programs. Our base language is CLU [14, 15]. Although the primitives are mostly independent of the base language, CLU is a good choice for two main reasons. It supports the construction of well-structured programs through its abstraction mechanisms, especially data abstractions; it is reasonable to assume that distributed programs will require such mechanisms to keep their complexity under control. Secondly, CLU is an object-oriented language, in which programs are thought of as operating on long-lived objects, such as data bases and files; this view is well-suited to the applications of interest, e.g., banking systems, airline reservation systems, office automation.

Our work this year has concentrated on primitives in two main areas, namely, modularity and communication. Below we discuss a new linguistic construct intended to support modular construction of distributed programs, and provide an example illustrating its use. More information about our work can be found in [13].

For distributed programs, a modular unit is needed that

1. Can be used to model the tasks and subtasks being performed in a reasonably natural way.

2. Can be realized efficiently, i.e., gives the programmer a realistic model of the underlying architecture.

A major issue in point (2) is control of direct sharing of data. An object that is shared directly (i.e., many entities know its location in the distributed address space) is a problem for three reasons. The object can be a bottleneck because of the contention for its use. It is a storage management problem, since to deallocate it while avoiding dangling references requires detection and invalidation of all references to it. Finally, to coordinate its use correctly can lead to increased program complexity. The main conclusion that can be drawn from considering these problems is that a linguistic mechanism that encourages the programmer to think about controlling the direct sharing of data is desirable. Note that a synchronization mechanism such as a monitor [5, 11] helps with the synchronization problem but not with the other two, since the monitor itself is a shared datum.

1. Guardians

We provide a construct called a *guardian* to support modular distributed programs. A guardian is a local address space containing objects and processes. A *process* is the execution of a sequential program. *Objects* contain data; objects are manipulated (accessed and possibly modified) by processes. Examples of objects are integers, arrays, queues, documents (in an office automation system), bank accounts and procedures. Objects are strongly typed: they may be directly manipulated only by operations of their type. The types may be either built-in or user-defined.

A computation consists of one or many guardians. Within each guardian, the actual work is performed by one or many processes. The processes within a single guardian may share objects, and communicate with one another via these shared objects.

Processes in different guardians can communicate only by sending messages. (Message passing is discussed in [13].) Messages will contain the *values* of objects, e.g., "2" or "#176538 $173.72" (the value of a bank account object). An important restriction ensures that the address space of a guardian remains local: it is impossible to place the *address* of an object in a message. It is possible to send a *token* for an object in a message; a token is an external name for the object, which can be returned to the guardian that owns the object to request some manipulation of the object. (A token is a sealed capability that can be unsealed only by the creating guardian.) The system makes no guarantee that the object named by the token continues to exist; only the guardian can provide such a guarantee. Thus a guardian is entirely in charge of its address space, and storage management can be done locally for each guardian.

A guardian exists entirely at a single node of the underlying distributed system: its objects are all stored on the memory devices of this node and its processes run on the processors of the node. During the course of a computation, the population of guardians will vary; new guardians will be created, and existing guardians may self-destruct. The node at which a guardian is created is the node where it will exist for its lifetime. It must have been created by (a process in) a guardian at that node. Each node comes into existence with a *primal* guardian, which can (among other things) create guardians at its node in response to messages arriving from guardians at other nodes. This restriction on creation of new guardians helps preserve the autonomy of the physical nodes.

A guardian is an abstraction of a physical node of the underlying network: it supports one or more processes (abstract processors) sharing private memory, and communicates with other guardians (abstract nodes) only by sending messages. In thinking about a distributed program, a programmer can conceive of it as a set of abstract nodes. Intra-guardian activity is local and inexpensive (since it all takes place at a single physical node); inter-guardian processing is likely to be more costly, but the possibility of this added expense is evident in the program structure. The programmer can control the placement of data and programs (a consequence of autonomy as discussed above) by creating guardians at appropriate nodes. Furthermore, each guardian acts as an autonomous unit, guarding its resource and responding to requests as it sees fit.

2. Robustness

A major problem in distributed programs is how to achieve robust execution of atomic operations in spite of failures. (An atomic operation is either entirely completed or not done at all.) This is an area where distributed programs are likely to differ significantly from centralized programs. Not that the need for robustness is new; rather, the issue has been largely ignored in centralized systems, with the exception of some work in data base systems.

One requirement for robustness is *permanence of effect*. Permanence means that the effect caused by a completed atomic operation (e.g., a change in the state of the resource owned by the guardian that performs the operation) will not be lost due to node failures.

To achieve permanence requires a finer grain of backup and recovery than is provided by occasional system dumps and automatic system restart. We believe that permanence must be provided by each guardian for the resource it guards. We expect that backup and recovery will be provided on a per guardian basis: processes in the guardian save recovery data as needed (by, e.g., logging it in storage that will survive a node crash), and the guardian provides a recovery process that is started after a node crash to interpret the recovery data.

3. Discussion and Examples

The guardian construct was invented to satisfy the modularity criteria given above. The purpose of a guardian is to provide a service on a resource of a distributed program, but in a safe manner, i.e., it guards the resource by properly coordinating accesses to it, by protecting the resource from unauthorized access, and by providing backup and recovery for the resource in case of node failures. The resources being so guarded may be data, devices or computation.

For example, the flight data for an airline might be guarded by a single guardian that handles reservations for all flights and also provides a number of administrative functions such as deleting or archiving information about flights that have occurred, collecting statistics about flight usage, etc. It responds to requests such as "reserve," "cancel," "list passengers," and so on. For such requests, it checks that the requestor has the right to request the access (perhaps using some sort of access control list mechanism [17]). For example, only a manager can request a passenger list, and a reservation request from some other airline might not be permitted to reserve the last seat on a flight. The guardian guarantees that requests are properly coordinated, for example, performed in an order approximating the externally observable order in which they were requested. It performs the reserve and cancel requests as atomic operations, and logs them so that information will not be lost if the node fails.

Internally, the airline guardian might make use of a guardian for each flight: the top level guardian simply dispatches requests to the appropriate flight guardian, which does the actual work and logs results. A flight guardian might be organized in several different ways, for example:

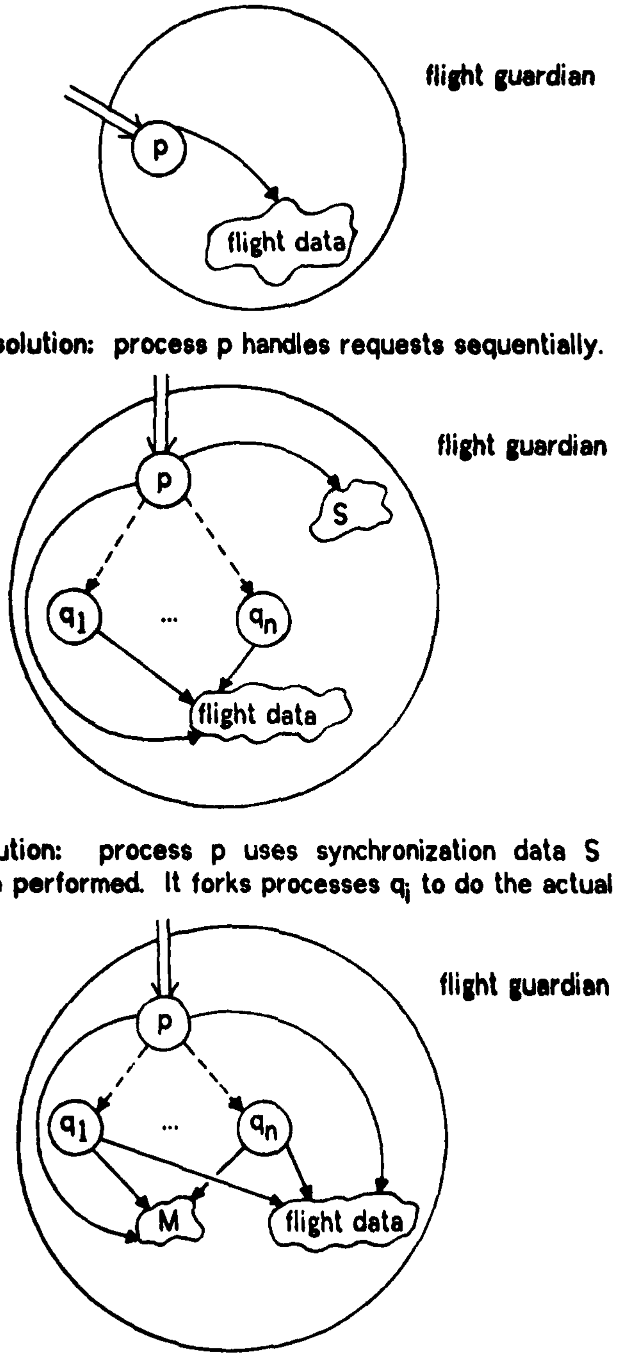
1. A single process handles requests one at a time (Figure 1a).

2. Requests for different dates are permitted to proceed in parallel. A single process synchronizes requests; it hands them off to other processes that perform the actual work (Figure 1b) when the flight data of interest are available. Such a structure is similar to that provided by a serializer [10].

3. A single process receives a request and immediately creates a process to handle it (Figure 1c). The forked processes synchronize with each other to ensure that only one process is manipulating the data for a particular date at a time. The processes synchronize using shared data, e.g., a monitor [11] providing operations *start_request(date)* and *end_request(date)*.

Organizations 2 and 3 can provide concurrent manipulation of the data base, while organization 1 cannot.

Fig. 1. Possible organizations for a flight guardian.



a. One-at-a-time solution: process p handles requests sequentially.

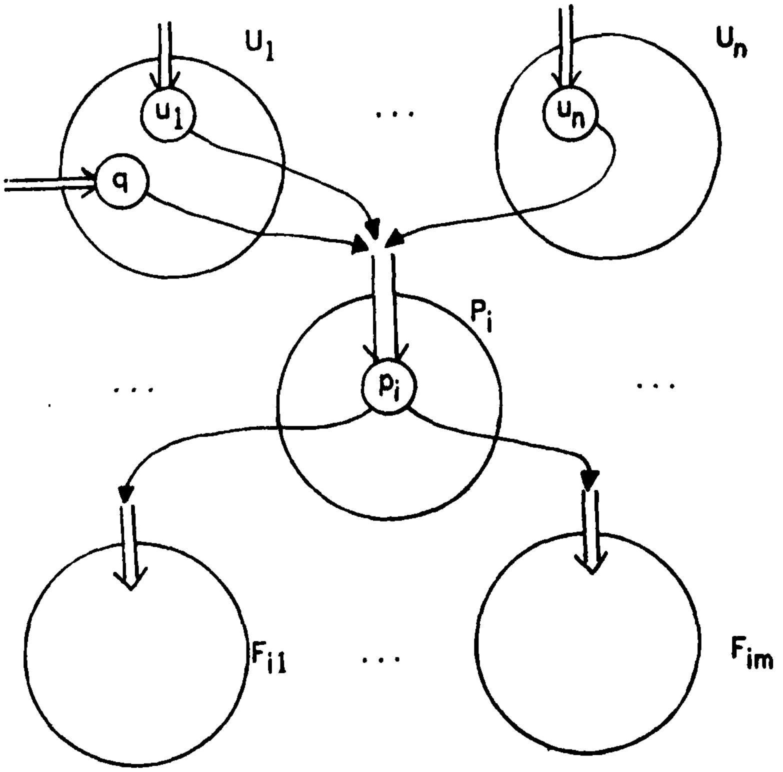
b. Serializer solution: process p uses synchronization data S to determine when requests should be performed. It forks processes q_i to do the actual requests.

c. Solution using a monitor: process p forks a process q_i upon receipt of a request. The processes q_i synchronize with each other using monitor M and perform the requests on the data base.

The airline data base discussed above had a single top level guardian. Alternatively the data base might be distributed; for example, it might be divided into partitions for different geographical regions, each residing at a distinct node, and the guardian for a flight assigned to the region containing the flight's destination. Such a structure is shown in Figure 2. Here each node belonging to the airline has one guardian, P_i, for the region in which it resides, and one guardian, U_i, to provide an interface to the airline data base for that node's users (e.g., reservation clerks and administrators). A user makes a request to the U_j at his node; some checking for access rights would be done here, and then one or more requests sent to the appropriate P_i. The P_i would dispatch these requests to the flight guardians for its region. The P_i and U_j would coordinate as needed by means of some protocol established for that purpose. A possible organization for the U_j might be to fork a process to handle a transaction

Fig. 2. Distributed airline system example.

There are n front ends (guardians U_1 to U_n) and n regional managers (only one, guardian P_i is shown) that communicate with the guardians of flights in its region (guardians F_{i1}, ..., F_{im}). Process q in U_1 is carrying out a transaction for a user. Processes u_i are ready to accept requests from new users.



consisting of many requests; this process would carry out U_j's end of the coordination protocol. This process might, for example, interact with a clerk to make a number of reservations for the same customer.

In the organization shown in Figure 2, each guardian U_j guards the entire airline data base and provides transactions that consist of sequences of requests. Each guardian P_i guards the data for a geographical region, while each flight guardian guards the data for a single flight. Thus, access to the entire distributed data base is provided by a group of guardians, but each guardian in that group guards a discernable resource.

It is appealing to imagine a system structure in which processes do not share any data. Although multi-process guardians are not necessary for computational power, we permit many processes in a guardian for two main reasons: concurrency (e.g., Figures 1b and 1c) and conversational continuity. Concurrency could be obtained by having guardians that guard very small resources (e.g., the information about a single flight and date, or a single record in a data base), but we felt such a structure would often be unnatural. Conversational continuity is illustrated in Figure 2: process q carries on a conversation with the user and the "state" of this conversation (e.g., the identity of the passenger for whom reservations are being made and the reservations made so far) is captured naturally in the state of process q.

D. NEW CLU IMPLEMENTATION

With the current CLU implementation in a fairly stable state, we have begun work on a new implementation for the LCS Advanced Node. This effort has four major goals: to produce a more efficient implementation, to produce a much more portable implementation, to provide a basis for implementing the extensions to CLU discussed in the preceding section, and to provide a modern programming system for laboratory-wide use.

To produce a more space-efficient implementation, we have decided to use a static linker and loader, at least initially, rather than copy the dynamic (re)linking and (re)loading capabilities of the current implementation [1]. Space is saved this way because the linker, loader, and associated data structures are not in memory at execution time. Although this decision eliminates a very useful debugging aid, it should be possible eventually to reintroduce some form of dynamic relinker/reloader running as a separate process.

In an attempt to produce a more time-efficient implementation, we have carefully redesigned the procedure call mechanism, the iterator mechanism, the parameter mechanism, and the exception handling mechanism. Whereas these mechanisms were designed for the current implementation primarily to allow simple, uniform code generation, for the new implementation we have tried to optimize the most common cases, without sacrificing speed in other cases. For the most part, these optimizations reduce the number of registers that must be maintained across procedure and iterator boundaries. In addition, we have decided not to maintain run-time type codes, which exist in the current implementation primarily for redundant run-time type-checking.

A major problem with the current implementation is that most of the run-time support system is written in assembly language. This not only complicates maintenance of the system, but also makes the system difficult to transport to a different machine, or even the same machine under a different operating system. We hope to rectify this problem with the new implementation. Although we have been proceeding under the assumption that the initial target machine is the Zilog Z-8000, it should also be possible to bring the system up on a VAX 11/780, an M68000, a 360/168, or a PDP-10 with a minimum of effort. The primary assumptions are that the machine supports values of at least 32 bits (for object references), a fairly large (2^{16} values or more) linear address space, an efficient stack mechanism for object references, and a small number of registers (for object references).

Easy portability is obtained by defining a small number of data types (object references, integers, tagged cells, byte vectors, and vectors of object references) that are not completely type-safe, and a few I/O primitives, and then implementing everything else in CLU on top of them. (With certain extensions to CLU, even records and oneofs can be implemented in CLU.) To bring up an implementation on a different machine, it should only be necessary to implement a few modules in assembly language, produce a new code generator for the compiler, and perhaps modify a few code-dependent modules of the linker.

The major components of the new implementation that need to be constructed are the basic data types of CLU, the garbage collector, the linker and loader, the code generator, and various debugging tools. All of these components have been designed and, with the exception of the debugging tools, are partially implemented and tested for the Z-8000.

E. EVALUATING SYNCHRONIZATION CONSTRUCTS

When facilities for concurrent programming are added to a language, it is essential that these extensions support the construction of reliable, easily maintained software. Much attention has been given to developing high-level programming language synchronization constructs, such as monitors [11], path expressions [6], and serializers [10], for specifying and controlling access to shared resources. Unfortunately, the requirements that these constructs must meet are not fully understood. Properties such as expressive power, ease of use, modularity, and modifiability are agreed to be important, but the definitions of these terms are so vague that evaluation according to these criteria is difficult.

Most attempts at evaluating these constructs have centered around attempting to implement solutions to various examples of synchronization problems, and then deriving intuitive judgments about the expressiveness and usability of a construct. Bloom [3] has developed evaluation methods that allow such information to be deduced from examples in more structured ways, and has applied the method to several existing mechanisms. First, synchronization problems have been examined and categorized according to properties that affect how easily these problems can be handled by synchronization constructs. This categorization defines the range of problems synchronization mechanisms must be able to handle, and provides a means of selecting a set of test cases adequate for evaluating the mechanisms. Making use of these test cases, mechanisms can be evaluated, with the methods developed, for expressive power, ease of use, and modifiability. Because the properties of interest are so vague, completely objective methods of evaluation are impossible. However, the categorization makes it possible to obtain information from test cases that will provide substantial assistance in locating and correcting weaknesses, as well as in generalizing results from one test case to sets of problems with similar properties.

In the following sections, we first discuss modularity requirements. Then the categorization of synchronization problems is presented, followed by an explanation of techniques for evaluating synchronization constructs based on this categorization. These techniques have been applied to monitors, path expressions, and serializers, and a summary of this evaluation is given in the final section.

F. MODULARITY REQUIREMENTS

The first property to be defined is modularity, and how modularity applies to the structure of shared resources; the remainder of the analysis assumes shared resources are properly modularized. The model of shared resources assumed here is based on the use of abstract data types [15]. Resources are considered to be objects of abstract types. A resource will therefore have a set of operations associated with it, and access to the resource will be possible only by invoking one of those operations.

There are two modularity requirements that should be satisfied by concurrent programs accessing shared resources. The first follows from the principle that the definition of an abstraction should be made independent of its use. The shared resource abstraction should thus contain all of the syncnronization code, as well as the resource definition. This structure will guarantee that users of the resource can assume it to be properly synchronized; no synchronization code need be located at each point of access to the resource.

The other modularity requirement governs the structure of the shared resource definition. The module implementing the shared resource serves two purposes: to define the abstract behavior of the resource, independent of whether concurrent access is allowed, and to provide the synchronization for controlling access. These two parts actually serve different functions and should be separable into two subsidiary abstractions, the unsynchronized resource, and the synchronization.

1. Categorizing Synchronization Problems

Synchronization mechanisms serve two main functions with respect to shared resources. One is excluding certain processes from the resource under given circumstances; the other is scheduling access to the resource according to given priorities. Synchronization schemes are thus composed of two types of constraints. Exclusion constraints take the form:

if condition then process A is excluded from the resource

and priority constraints take the form:

if condition then process A has priority over process B

Within these two main classes, constraints differ in the kinds of information used in the conditional clause. The information that can appear falls into several categories:

1. The access operation requested. Stating, for instance, that readers of a data base have priority over writers involves giving a constraint in terms of the types of operations requested. In contrast, a strict first-come first-serve ordering uses no information about the operations requested.

2. The times at which requests were made. Though it is rarely necessary to know the exact time of a request, the order of requests relative to other events is often important. Time information is used for this purpose.

3. Request arguments. In many cases, the arguments passed with a request are needed to determine the order in which processes should be admitted to the resource.

4. Local resource state. Local state includes information that would be present independent of access control. For example, whether a buffer is full or empty.

5. Synchronization state. Synchronization state includes only state information needed for synchronization purposes. This information would not be part of the resource state were the resource not being accessed concurrently. Included in this category is information about the processes currently accessing the resource, and the operations those processes are executing.

6. History information. This information differs from synchronization state in that it refers to resource operations that have already been completed, as opposed to those still in progress. It is often interchangeable with local state information, since the interesting past events are most likely to be those that leave some noticeable change in the state of the resource.

By way of two simple examples, a first-come first-serve ordering scheme has a single priority constraint of the form:

```
process A has priority over process B
        if time_of_request(A) < time_of_request(B)
```

Thus, only request time information is used. A readers priority constraint has the form:

```
process A has priority over process B
        if request_type(A) = read & request_type(B) = write
```

More complete examples can be found in [3].

This identification of properties of synchronization problems enables more informative conclusions to be drawn from the evaluation of synchronization problems. In addition, the categorization aids in the selection of test cases. By selecting a set of problems that cover all the constraint types, and examining the ways in which each kind of information is handled by a particular mechanism, general conclusions can be drawn about the mechanisms' ability to implement synchronization problems that make use of various types of information.

2. Evaluation Criteria

To be sufficiently powerful, a synchronization mechanism must satisfy two basic requirements. First, it must be expressive, by providing a straightforward means of stating individual constraints, and by providing the ability to express those constraints in terms of any of the information types described earlier. Second, when implementing complex synchronization schemes composed of many constraints, it must be possible to implement each constraint independently. In the following sections we explain these criteria more precisely, and discuss methods for assessing how well they are supported by various synchronization constructs.

2.a. Expressive Power

One way to test expressive power of a synchronization mechanism is to use it to implement solutions to a set of examples that cover all information classes. A sample set is presented in [3]. If there is no direct way to use a certain kind of information, it should become obvious when an attempt is made to implement a solution requiring it. By examining how various types of information are handled in each solution, conclusions can be drawn as to the ease with which the mechanism can access each type of information.

A more general way to measure expressive power is simply to examine the mechanism and attempt to determine what features it has that will enable each type of constraint to be dealt with. For example, monitor queues are a construct for handling request time information, while serializer crowds retain synchronization state information. Some data manipulation technique must be available for each type of information. The ability to identify the particular way in which each information type is handled will also make a mechanism easier to use, because the structure of a solution will be indicated by the kinds of information used in the specification.

2.b. Combining Constraints in Complex Solutions

Whether or not a mechanism is easy to use depends not only on the ability to easily construct solutions to individual constraints, but on the ability to easily construct complex synchronization schemes made up of many such constraints.

Complex schemes will be easy to implement only if they can be decomposed into individual constraints that can then be realized independently. As the number of constraints increases, solutions quickly become difficult to construct if the implementation of any one constraint depends on another; the complexity of constructing the solution increases more than linearly with the number of combinations of constraints present. In addition to the difficulty of initially constructing solutions, the solutions will be difficult to modify if this constraint independence property is not met: a change in the specification of one constraint may necessitate reimplementation of the entire solution.

One way to test whether a mechanism allows independent implementation of constraints is to examine solutions to two similar synchronization problems. If the problems share some constraints, but differ in others, then the implementation of the common constraints should be similar in both solutions. If differences appear in the way a given constraint is implemented in two different synchronization problems, or if the implementation of each individual constraint is not even identifiable as a separate part of the solution, then this indicates that the independence criterion is being violated. If constraint implementations are really independent, a given implementation should remain the same when other constraints are modified to use different types of information. A complete evaluation involves checking all possible pairs of the six information types for conflicts. Two sample problems for use in this analysis are given in [3].

3. Evaluation of Existing Mechanisms

The three existing mechanisms that seem most likely to satisfy the requirements of good software engineering are monitors, path expressions, and serializers. Based on our evaluation, we have drawn the following conclusions about these three mechanisms. While the approach taken by path expressions seems very attractive, our analysis has revealed some serious shortcomings. Path expressions do not provide easy access to several types of information needed in synchronization constraints, and thus lack sufficient expressive power. In particular, it is difficult to use the local resource state and the arguments of operations. To maintain information about time of request, or to express priority constraints in general, requires additional synchronization procedures, thus increasing the solution's complexity. In addition, the constraint independence requirements necessary to ensure ease of use are not well supported by the mechanism.

Both monitors and serializers satisfy our criteria reasonably well. Based on our evaluation, we prefer serializers over monitors. Though certain tradeoffs are involved in selecting one of these mechanisms over the other, serializers seem superior in two important respects. First, serializers meet our modularity requirements more closely. The proper use of monitors requires a special protected-resource module in addition to the synchronizer and resource modules; the resource implementor must also follow specific guidelines for defining monitor operations. Serializers depend less on such rules: the protected-resource module is not needed, and serializer operations are precisely the user-accessible operations on the protected resource. The other important distinction between these two mechanisms is the use of automatic signalling in serializers. Though proof rules for the monitor signal construct have been developed, an automatic signalling feature is more likely to aid in constructing correct programs, and in easing the burden placed on the verifier.

G. CLU MACHINE ARCHITECTURE

The design of computers is strongly influenced by the characteristics of available technology. Until recently, computers have been designed under the constraint that processing hardware is expensive. However, the cost of hardware is continually decreasing and the significant cost of software has become even more apparent. Therefore, it seems appropriate to consider how hardware technology can be used to implement modern programming languages and to reduce the complexity of computer systems.

Snyder [19] has designed a computer system that directly supports an object-oriented machine language. Unlike most implementations of object-oriented languages on conventional machines, which provide a separate and usually small space of objects for each process, the proposed system provides a single, very large space of objects shared by all processes in the system. This space of objects would include not only temporary objects used during the execution of programs, but also such long-lived objects as the procedures and data normally stored in a file system, with uniform access to all objects.

The primary goal of the research was to design a machine that supports such a large universe of objects effectively. A second goal was to minimize the complexity of the design. To accomplish these goals, two assumptions were made about future technology.

The first assumption is that processors are sufficiently inexpensive that several processors can be used where one is used today. Multiple processors are used both to obtain greater modularity, and to improve performance. When necessary, processor utilization may be decreased in order to increase system throughput.

The second assumption is the existence of secondary storage devices with access times on the order of 100 microseconds (about 100 times faster than current disks). This assumption is motivated by the expected small average size of objects (on the order of 4 to 20 words), based on measurements of existing programs Fast-access devices are needed to obtain good multi-level memory performance without introducing undue complexity.

Both of these assumptions appear to be reasonable. It is widely predicted that LSI processors equivalent to current main frames will be developed in the next decade; the cost of these processors will be quite low compared to the total cost of a computer system. The access time figure of 100 microseconds is within the predicted range for charge-coupled devices and electronic beam memories; however, projections do show that such memories may be an order of magnitude more expensive than disk memories.

Below we provide a brief description of this work. The first section focuses on the overall system structure, and the second section explains the storage reclamation mechanism.

1. System Structure

The system is constructed hierarchically out of a number of specialized processor modules communicating via messages. Each module performs well-defined functions. At the top-most level, the system is divided into two major modules, the processing module (PM) and the memory module (MM). The PM interprets procedures and implements multiple processes. It consists of a number of instruction processors, which interpret procedures, plus a control processor, which performs scheduling and controls the multiplexing of the instruction processors. Each instruction processor has its own local memory which it uses in the interpretation of instructions. In addition, this local memory is used as a cache to reduce the number of accesses to the MM. For example, much of the evaluation stack would certainly be held in this local memory.

The MM implements the universe of objects with a multi-level memory system. An object is simply a vector of references to other objects (although certain values, such as booleans and integers, are stored directly in a reference). The MM encapsulates all knowledge of how objects are implemented, including storage allocation and automatic storage reclamation. The interface between the PM and the MM basically consists of invocations of operations of the vector data type.

Briefly, each object is represented by a single "page"; the system supports a number of different page sizes. Objects (pages) are identified by their secondary storage addresses and are transferred individually between primary and secondary storage. A large set associative memory maps from the secondary storage addresses of objects in primary storage to their primary storage addresses. The set associative memory is implemented using ordinary random access memory, with a small, fast, expensive associative memory used as a cache. Physical storage is divided into fixed-size blocks; each block is (statically or dynamically) divided into pages of a single size. Storage is further divided into a number of zones; each zone provides pages of a single size and contains its own list of free pages.

2. Storage Reclamation

To maximize secondary storage utilization, the storage used by an object should be reclaimed as soon as possible after the object becomes inaccessible. An important contribution of this research has been the development of a simple, efficient automatic storage reclamation scheme that is performed continuously, without requiring frequent or unpredictable interruptions of service.

To facilitate storage reclamation, there is one additional interaction between the PM and MM. The PM must cooperate with the MM in order for the MM to determine which objects are needed and which can be reclaimed. In particular, at certain times the MM will request the PM to discard all of its object references. When there are no object references outside the MM, the system is said to be in *quiescence*. During quiescence, the MM can examine the entire collection of accessible objects without interference from the PM. When the MM is finished, the PM reads back all needed data and resumes normal operation.

The storage reclamation algorithm is based on reference counts. The basic idea of reference counts is to associate with each object a counter to record the number of existing references to that object. When an object is created, a single reference to the object is created, and the reference count of the object is set to one. Whenever a reference to the object is copied or destroyed, the reference count is incremented or decremented, respectively. The object can be reclaimed whenever the reference count reaches zero (destroying all contained object references and decrementing the corresponding reference counts). Of course, if a group of objects contains a cycle of references, none of the objects in the group can be reclaimed in this way, even if the entire group is inaccessible. Similarly, if a bounded reference count ever reaches its maximum value, it must remain there, lest the object be reclaimed prematurely.

The biggest problem with a conventional reference count scheme is that reference count events occur at an enormous rate: each time a reference to a storage object is copied or destroyed, a reference count must be updated. A reference count scheme can be made very efficient by not requiring every reference in the system to be accounted for. Instead, reference counts only count references stored as components of objects in the MM; references outside the MM, or on their way in or out of the MM, are not counted. This substantially reduces the number of events that cause reference count operations: only events that change the contents of objects in the MM cause reference count operations. Manipulations of references within the local memory of the PM do not change reference counts. Since most of a process' evaluation stack can be cached into such local memory, changes to the evaluation stack generally cause no reference count operations.

Of course, under this scheme an object cannot be reclaimed just because its reference count has become zero; rather, the system must be forced into quiescence in order to reclaim objects. In quiescence, one can locate objects with zero reference counts by scanning the entire memory, but a much more efficient algorithm is possible.

During normal operation (not during quiescence), whenever the reference count of an object X becomes zero, an entry "discard(X)" is added to a queue of suspected garbage GQ. Further, whenever the reference count goes from zero to non-zero, an entry "resurrect(X)" is added to the GQ. Then, when quiescence is established, the GQ can be used to determine precisely which objects can be reclaimed: an object X can be reclaimed only if the last entry for it on the GQ is of the form "discard(X)". This information is totally contained within the GQ; there is no need to examine the actual reference counts of objects. More importantly, it is not necessary to keep the system in quiescence while the GQ is being processed. Once quiescence is established, normal system operation can be resumed with a new, empty GQ, with the old GQ being processed concurrently.

Forcing the system into quiescence is similar to swapping out all running processes. In general, there will be some minimum rate of process switching needed anyway to maintain interactive response with the users. As long as the cycle time between quiescent states is longer than the process time quantum, there need be little performance degradation.

In addition to the storage reclamation process just described, it will be necessary to perform infrequent, periodic garbage collection for the purpose of reclaiming cyclic garbage. Suitably designed, the garbage collection time should be on the order of ten minutes [19]. Clearly any time of this magnitude is acceptable for infrequent garbage collections scheduled on the order of once per day or once per week.

H. SPECIFICATIONS OF MUTABLE ABSTRACTIONS

Valdis Berzins [2] has investigated specifications for data abstractions based on the *explicit* or *abstract model* appproach. In this technique, a data abstraction is defined in terms of simple, well-known abstractions (e.g., integers, mathematical sets), and possibly some other user-defined abstractions. The main innovation of this work is that it extends previous work to handle both potentially shared, mutable data objects, and operations that can raise exceptional conditions.

In the sections below, we give an informal description of this work, concentrating on the extension to mutable objects. We first give an example specification of a mutable data type, integer sets, to present the specification language and explain the major concepts of the technique. Following the example, we discuss the standard model defined by a specification. We then present an implementation of integer sets, and show how to prove that an implementation model is equivalent to the standard model. We conclude with a discussion of how the approach fits in with program verification.

1. Example Specification

A specification of finite, mutable sets of integers is shown in Figure 3. The first line of the specification states that we are defining a type named "intset," and introduces "I" as an abbreviation for "intset." The with clause gives the names and functionality of the operations. For example, "empty" takes no arguments and returns an *intset*, and "remove" takes two arguments and returns no results (it mutates its first argument).

Because we are specifying a mutable data type, the actual model is based on *system states*, which are indexed sets of *data states*. Each data state represents the state of some *intset* object. Formally, if s is a system state and x is an *intset token* (a formal identifier for an *intset* object), then $s(x)$ is the state of the particular *intset* object x in the system state s. The actual functionality of each operation has an extra argument (the input system state) and an extra result (the output system state) besides those shown in the with clause of the specification. The definitions of immutable data abstractions are, of course, simpler.

As types are built up in a hierarchy, the system state becomes a set of indexed sets, with one indexed set of data states for each mutable type used in the type's operations. The formal structure that is built is a heterogeneous algebra.

The data states for *intset* are represented by mathematical sets of integers; the restrictions clause indicates that only finite sets can be used. The identity clause defines an *equivalence relation on the restricted data state domain; the objects of the*

Fig. 3. Specification for Intset

```
type intset as I

with    empty:                 --> I
        insert:     I x int    -->
        remove:     I x int    -->
        has:        I x int    --> bool

data states   D = sets of integers
restrictions  d such that cardinality (d) ∈ N
identity      equal

operations    empty(s)( ) = extend (s, { })
              insert(s) (x, i) = update (s, x, s(x) ∪ {i})
              remove (s) (x, i) = update (s, x, s(x) ∪ {i})
              has (s) (x, i) = <s, i ∈ s(x)>

end intset
```

algebra are precisely the equivalence classes so formed. In this case the identity is trivial (set equality).

The operations clause of the specification defines the effects of the operations. Here the system state is represented explicitly. Also, to emphasize the special nature of the system state, the operations are written in "curried" form; e.g., we write "insert(s)(x, i)" instead of "insert(s, x, i)". The operations are defined in terms of functions on the system state: "extend" creates a new object with the given initial data state, and returns both a system state with the new object added and a token for that new object; "update" returns a new system state with the data state of the object named by its second argument changed to be the data state given as the third argument.

2. Standard Model

So far the presentation has been largely intuitive; we have not said much about the formal object that is defined by a specification. To the logician, a model is a mathematical structure satisfying some set of axioms; the existence of a model shows the set of axioms to be consistent. A specification can also be viewed as a set of axioms; the heterogeneous algebra defined by the specification is a model for that axiom system. This model is called the *standard model.*

Since the specification language is powerful enough to describe inconsistent specifications (any useful language has this property), we must, in general, prove the existence of a standard model. This is done by proving inductively that no operation constructs a data state outside the restricted data state domain (when provided with arguments in that domain), and that all operations preserve the equivalence relation on data states (i.e., equivalent arguments produce equivalent results). In our example, the proof is trivial, but in more complicated types this may require considerable effort.

3. Example Implementation

The specification language also has features for defining implementations. Figure 4 gives an implementation of *intset* in terms of arrays of integers.

The representation clause serves both to identify what type is being implemented, and to describe the representation domain. Arrays are mutable vectors to which items can be added or removed at either end, and they are indexed by a contiguous subset of the integers. For an array *a*, low(*a*) is its current low bound, high(*a*) is its current high bound, and if low(*a*) $\leq i \leq$ high(*a*), then *a*[*i*] names a valid element of *a*.

The restrictions clause states which elements of the representation domain are legal representations of objects. Here the arrays are restricted to have a low bound of 1, and to not have any duplicate elements. The identity clause states that identity of an *intset* object is to be represented by the identity of the representing array. Note that this clause defines object identity, *not* state identity; two distinct objects may have the

Fig. 4. An Implementation of Intset

```
representation   intset = array[int]

restrictions     a such that: low(a) =  1 &  (low(a) ≤ j < k ≤ high(a) ==> a(j) ≠ a(k))
identity         array$equal

operations       empty( ) = array[int]$create(1)
                 insert (a, i) = if ~has (a, i) then addh (a, i)
                 remove (a, i) = if has (a, i) then {store (a, find(a, i), a[high(a)]); remh(a)}
                 has (a, i) = find(a, i) > 0

definition  find(a, i) = if (∃j) [low(a) ≤ j ≤ high(a)  &  a[j] = i]
                                     then j : low(a) ≤ j ≤ high(a)  &  a[j] = i
                                     else 0
```

same state. For mutable objects, identity is almost always identity of representation objects, as it is in this case. Identity for immutable objects can be more complicated, particularly if the representataion is itself mutable.

The **operations** clause defines the *operations in terms of the representation* domain; each operation is defined by a program in a simple programming language. This language permits ∃, ∀, and ":" (meaning "such that") to be used for clarity and to avoid overspecification. The operations are implemented in terms of array operations: "create" returns a new, empty array with the given low bound; "addh" adds an element to the high end of an array; "remh" removes and returns the high element; "store" updates the element at the specified index with the given value. Some of these operations may raise exceptional conditions; in this example it must be proved that they do not. The **definition** clause is used for defining "helping routines" to simplify the description of the implementation.

4. Behavioral Equivalence

An implementation defines a model, in a manner very similar to the way a specification defines the standard model. To show that we have a valid implementation of an abstraction, we must demonstrate that the implementation simulates the required abstract behavior. The particular simulation desired is *behavioral equivalence*: informally, this means that if the implementation model is substituted for the standard model in any program, the program will produce the same results.

Behavioral equivalence is proved inductively on the length of the computations performed by the two programs (one with the standard model, one with the implementation model). At each step in the computation we must exhibit a correspondence relation between the simulated objects and the simulating ones. It is this relation that describes exactly how the implementation objects simulate the abstract objects. As such, its purpose is very similar to Hoare's abstraction function [12].

The simulation relation for our example is as follows, where "abstract" refers to the standard model and "concrete" refers to the implementation model:

For each	i	abstract integer,
	i', j'	concrete integer,
	s	intset abstract system state,
	s'	intset concrete system state,
	x	abstract intset,
	x'	concrete intset

$$\begin{aligned} & s \Longleftrightarrow s' \ \& \ x \Longleftrightarrow x' \ \& \ i \Longleftrightarrow i') \Longrightarrow \\ & \qquad (i \in s(x) \ \equiv \ (\exists j')\,[\mathrm{low}\,(s'(x')) \le j' \le \mathrm{high}(s'(x')) \ \& \ i' = s'(x')\,[j']]) \end{aligned}$$

This says that given corresponding system states (s, s'), *intset* objects (x, x'), and integers (i, i') in the two models, then an integer is an element of the *intset* in one model if and only if the corresponding integer is an element of the corresponding *intset* in the other model.

5. Program Verification

Proving the correctness of a data type implementation with respect to a standard model is only half of the process required to verify programs that use data abstractions. The other half of the process involves proving the correctness of programs that invoke the type's operations. The intended behavior of a program is typically described by inserting assertions at various points in the program. The assertions express the relations that must hold between the data objects manipulated by the program. For programs that use data abstractions, the assertions are written in terms of the primitive operations of the abstractions. For dynamic abstractions, the system state must be explicitly included in the assertions, so that the operations can be treated as pure functions.

The problem of showing that a program satisfies its assertions can be reduced to the problem of proving theorems about the data abstractions it uses, by eliminating the program text from the correctness requirements with an axiomatic definition of the control constructs in the programming language. The derived theorems, which must be proved in order to establish correctness, are called verification conditions. Proving verification conditions based on an abstract model approach presents no methodological problems. It is sufficient to prove the interpretations of the verification conditions in the standard models of the data abstractions used by the program, since behavioral equivalence guarantees the same results in all correct implementations of those abstractions.

References

1. Atkinson, Russell R., Liskov, Barbara, and Scheifler, Robert W. "Aspects of implementing CLU," Proceedings of the ACM 1978 Annual Conference, Washington, D.C., December 1978, 123-129.

2. Berzins, Valdis A. "Abstract model specifications for data abstractions," MIT/LCS/TR-221 MIT, Laboratory for Computer Science, Cambridge, Ma., July 1979.

3. Bloom, Toby. "Synchronization mechanisms for modular programming languages," MIT/LCS/TR-211, MIT, Laboratory for Computer Science, Cambridge, Ma., April 1979.

4. Brinch Hansen, Per, "Distributed processes: A concurrent programming concept," Communications ACM 21, 11 (November 1978), 934-941.

5. Brinch Hansen, Per, "The programming language concurrent Pascal," IEEE Transactions on Software Engineering 1, 2 (June 1975), 199-207.

6. Campbell, Roy H. and Habermann, A. Nico, "The specification of process synchronization by path expressions," in Operating Systems, Lecture Notes in Computer Science, Vol. 16, Springer-Verlag, 1974.

7. Feldman, Jerome A. "A programming methodology for distributed computing," University of Rochester, Department of Computer Science, Technical Report 9, Rochester, N. Y., 1977.

8. Fuller, Samuel H., et al. "A collection of papers on CM*: A multi-microprocessor computer system," Carnegie-Mellon University, Department of Computer Science, February 1977.

9. Hewitt, Carl E. "Viewing control structures as patterns of passing messages," Artificial Intelligence 8, 3 (June 1977), 323-364.

10. Hewitt, Carl E. and Atkinson, Russell R. "Specification and proof techniques for serializers," IEEE Transactions on Software Engineering SE-5, 1 (January 1979), 10-23.

11. Hoare, C. A. R. "Monitors: An operating system structuring concept," Communications ACM 17, 10 (October 1974), 549-557.

12. Hoare, C. A. R. "Proof of correctness of data representations," Acta Informatica 1, 4 (1972), 271-281.

13. Liskov, Barbara, "Primitives for distributed computing," MIT, Laboratory for Computer Science, Computation Structures Group Memo 175, Cambridge, Ma., May 1979.

14. Liskov, Barbara, Moss, Eliot, Schaffert, J. Craig,, Scheifler, Robert W., and Snyder, Alan, "CLU reference manual," MIT, Laboratory for Computer Science, Computation Structures Group Memo 161, Cambridge, Ma., July 1978.

15. Liskov, Barbara, Snyder, Alan, Atkinson, Russell, and Schaffert, J. Craig. "Abstraction mechanisms in CLU," Communications ACM 20, 8 (August 1977), 564-576.

16. Peterson, James L. "Notes on a Workshop on Distributed Computing," Operating Systems Review 13, 3 (July 1979), 18-30.

17. Saltzer, Jerome H. and Schroeder, Michael D. "The protection of information in computer systems," Proceedings of the IEEE 63, 9 (September 1975), 1278-1308.

18. Scheifler, Robert W. "An analysis of inline substitution for a structured programming language," Communications ACM 20, 9 (Sept. 1977), 647-654.

19. Snyder, A. "A machine architecture to support an object-oriented language," MIT/LCS/TR-209, MIT, Laboratory for Computer Science, Cambridge, Ma., March 1979.

Publications

1. Atkinson, Russell R., Liskov, Barbara H., and Scheifler, Robert W. "Aspects of implementing CLU," Proceedings of the ACM 1978 Annual Conference, Washington, D.C., December 1978, 123-129.

2. Bloom, T. "Synchronization mechanisms for modular programming languages," MIT/LCS/TR-211, MIT, Laboratory for Computer Science, Cambridge, Ma., April 1979.

3. Clark, David C., Greif, I., Liskov, Barbara H., and Svobodova, L. "Semantics of distributed computing, progress report of the Distributed Systems Group," MIT, Laboratory for Computer Science, Computation Structures Group Memo 171, Cambridge, Ma., October 1978.

4. Greif, I. and Meyer, A. "Specifying programming language semantics: A tutorial and critique of a paper by Hoare and Lauer," Proceedings of the Principles of Programming Languages, San Antonio, Tx., January 1979, 180-189.

5. Greif, I. and Meyer, A. "Specifying the semantics of while-programs: A tutorial and critique of a paper by Hoare and Lauer," MIT/LCS/TM-130, MIT, Laboratory for Computer Science, Cambridge, Ma., April 1979.

6. Kapur, D. "Specifications of Majster's traversable stack and Veloso's traversable stack," SIGPLAN Notices 14, 5 (May 1979), 46-53.

7. Kapur, D. and Mandayam, S. "Expressiveness of the operation set of a data abstraction," MIT, Laboratory for Computer Science, Computation Structures Group Memo 179, Cambridge, Ma., June 1979.

8. Laventhal, Mark S. "Synthesis of synchronization code for data abstractions," MIT/LCS/TR-203, MIT, Laboratory for Computer Science, Cambridge, Ma., July 1978.

9. Liskov, Barbara H., Moss, Eliot, Schaffert, J. Craig, Scheifler, Robert W. and Snyder, Alan, "CLU reference manual," MIT, Laboratory for Computer Science, Computation Structures Group Memo 161, Cambridge, Ma., July 1978.

10. Liskov, Barbara H. "Practical benefits of research in programming methodology," MIT, Laboratory for Computer Science, Computation Structures Group Memo 166, Cambridge, Ma., August 1978.

11. Liskov, Barbara H. "Primitives for distributed computing," MIT, Laboratory for Computer Science, Computation Structures Group Memo 175, Cambridge, Ma., May 1979.

12. Liskov, Barbara H. "Remarks on the construction of large programs," The Impact of Research on Software Technology, Wegner, P. (Ed.), MIT Press, Cambridge, Ma., June 1979, 345-351.

13. Liskov, Barbara H. and Berzins, V. "An appraisal of program specifications," The Impact of Research on Software Technology, Wegner, P. (Ed.), MIT Press, Cambridge, Ma., June 1979, 276-301.

14. Principato, Robert N., Jr. "A formalization of the state machine specification technique," MIT/LCS/TR-202, MIT, Laboratory for Computer Science, Cambridge, Ma., July 1979.

15. Snyder, A. "A machine architecture to support an object-oriented language," MIT/LCS/TR-209, MIT, Laboratory for Computer Science, Cambridge, Ma., March 1979.

16. Svobodova, Liba, Liskov, Barbara, and Clark, David, "Distributed computer systems: Structure and semantics," MIT/LCS/TR-215, MIT, Laboratory for Computer Science, Cambridge, Ma., April 1979.

Accepted for Publication

1. Liskov, Barbara H. and Snyder, A. "Exception handling in CLU," to be published in the *IEEE Transactions on Software Engineering*.

2. Peterson, James L. "Notes on a Workshop on Distributed Computing," to be published in Operating Systems Review.

Theses in Progress

1. Allen, Matthew D. "A comparative analysis of programmming languages," S.B. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected September 1979.

2. Atkinson, Russell R. "Verification of serializers," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected June 1980.

3. Berzins, V. "Abstract model specification for data abstractions," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected July 1979.

4. Herlihy, M. "Transmitting abstract values in messages," S.M. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected June 1980.

5. Kapur, D. "Towards a theory of data abstractions," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected June 1980.

6. Knopf, R. "A formatter for CLU," S.B. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected September 1979.

7. Leach, P. "Objects and information containers in a CLU garbage collector," S.B. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected January 1980.

8. Moss, E. "Distributed programming environment," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected June 1980.

9. Mullendore, C. "Performance analysis of the CLU implementation," S.B. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected January 1980.

10. Schaffert, J. C. "Specifications and proofs in object oriented languages," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected June 1980.

11. Srivas, M. K. "Automatic generation of implementations of data abstractions," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected June 1980.

Theses Completed

1. Bloom, T. "Synchronization mechanisms for modular programming languages," S.M. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., April 1979.

2. Snyder, A. "A machine architecture to support an object-oriented language," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., March 1979.

3. Zachary, J. "A CLU machine design evaluation," S.B. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., May 1979.

Talks

1. Greif, Irene, "Specifying the semantics of while-programs,"
   University of Washington, Computer Science Department, Seattle, Wa., January 1979;
   IBM, Yorktown Heights, N. Y., March 1979.

2. Liskov, Barbara H. "Issues in distributed computing," Quality Software Workshop, Salt Lake City, Utah, October 1978.

3. Liskov, Barbara H. "Implementation aspects of CLU," ACM National Conference, Washington, D.C., December 1978.

4. Liskov, Barbara H. "Use of data abstractions in data bases," ACM National Conference, Washington, D. C., December 1978.

5. Liskov, Barbara H. "Linguistic support for distributed computing," Eidgenossische Technische Hochschule, Zurich, Switzerland, January 1979.

6. Liskov, Barbara H. "Introduction to CLU," Copenhagen Winter School on Abstract Software Specifications, Copenhagen, Denmark, January 1979.

7. Liskov, Barbara H. "An example of modular program development," Copenhagen Winter School on Abstract Software Specifications, Copenhagen, Denmark, January 1979.

8. Liskov, Barbara H. "Embedding data abstraction in programming languages," Copenhagen Winter School on Abstract Software Specifications, Copenhagen, Denmark, January 1979.

9. Liskov, Barbara H. "Introduction to CLU," "An example of modular program development," Bell Laboratories, Piscataway, N. J., March 1979.

10. Liskov, Barbara H. "Linguistic support for distributed programs," University of Rochester, Rochester, N.Y., April 1979.

11. Liskov, Barbara H. "Message passing primitives," Quality Software Workshop, Amherst, Ma., April 1979.

12. Liskov, Barbara H. "Communicating abstract values," National Computer Conference, New York, N. Y., June 1979.

PROGRAMMING TECHNOLOGY GROUP

Academic Staff

A. Vezza, Group Leader

J. C. R. Licklider

Research Staff

M. S. Blank
M. S. Broos
S. W. Galley
D. S. Gerson
P. D. Lebling
S. S. Pinter
D. Sherry
M. I. Travers

Graduate Students

T. A. Anderson
S. N. Bhatt
W. J. Noss
W. G. Paseman

Undergraduate Students

G. C. Albinger
P. Arndt
S. H. Berez
D. L. Dill
A. Ghaznavi
D. A. Holt
T. K. Johnson
G. E. Kaiser
P. C. Lim
S. C. Phillips
C. A. Renaud
A. L. Ressler
B. J. Roberts
W. A. Seltzer
W. W. St. Clair

Support Staff

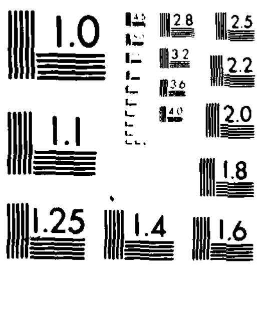
S. P. Briggs

J. L. Schoof

AD-A088 355 MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/6 9/2
LABORATORY FOR COMPUTER SCIENCE PROGRESS REPORT 16, 1 JULY 1978--ETC(U)
AUG 80 M L DERTOUZOS N00014-75-C-0661
UNCLASSIFIED LCS/PR-16 NL

2 OF 2
AD
A088355

END
DATE
FILMED
9-80
DTIC


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

A. INTRODUCTION

The Programming Technology group is engaged in two distinct research and development programs. The program in Morse code has as its main goals the development of the conceptual insight necessary to develop a computerized Morse-code operator and the design and implementation of a prototype of such a computer system (COMCO-I)[1]. The Morse-code program covers four areas: signal processing, Morse-code transcription, sender recognition, and understanding of the conversations among operators in a radio network that are carried on in a special language consisting of "Q-signs," "Pro-signs," and "Call-signs." The other research program, which has just begun, is concerned with the concepts, design and development of Data Intensive Planning Systems. Such systems are concerned with large-scale planning of processes that possess geographic characteristics involving many planners, a large volume of data and many variables.

B. MORSE CODE

COMCO-I, the prototype computerized Morse-code operator, is composed of four major subsystems.

1. A signal acquisition and processing module produces a file of mark (dot and dash) and space durations based on analysis of a signal.

2. A transcription module converts the mark and space durations into a lattice of possible transcriptions of the message, where each branch of the lattice is a vocabulary element from a large but finite vocabulary. This module begins by performing a MAUDE-like [1] assignment of each code element to its apparent type, and then passes that result to COMDEC (COmputerized Morse DECoder), which builds the lattice of transcriptions.

3. The transcriptions suggested by COMDEC are evaluated by CATNIP, a parser based on augmented transition networks, which attempts to derive coherent, "grammatical" transmissions from them.

4. Finally, MAGE (Morse Automatic Grammar Extension system) attempts to extend CATNIP's grammar in a rational and consistent manner if the lower level modular CATNIP and COMDEC cannot derive a coherent grammatical expression from the transmission.

1. Morse-Code Transcription

Development of COMDEC continued during the last year, but at a lower level of intensity than in previous years. This is primarily because COMDEC is now a "mature" program, and because the newer modules of COMCO-1 depend on its reliable operation.

Changes in COMDEC were (with the exception of the usual bug fixes) confined to the interaction between COMDEC and CATNIP, and more generally, on improving COMDEC's performance in handling the environment of Morse-code radio network "chatter".

The most important expansion of the COMDEC-CATNIP interface occurred in the domain of "vocabulary selection" (Lebling, Kaiser). That is, previously the mode of operation of the transcription was to have available a global "vocabulary" which contained all possible words in the transmission. The capability to divide this vocabulary into smaller vocabularies has existed for a long time, but the selection of those vocabularies was always an operation requiring user intervention.

Having the entire vocabulary available at all times has advantages (the likelihood of "missing" an unknown word is reduced, for example), but the disadvantages outweigh them. Specifically, in the environment of Morse-network "chatter", most of the interaction consists of short words, often irregularly spaced and badly sent. The larger the vocabulary, the larger the chance of confusing the transcription modules by accidental overlap of a given code sequence.

It was decided to select only those vocabularies expected to be useful in transcribing the area of code being worked on. The problem then was to change that selection when appropriate, specifically, when the code being transmitted contains English text.

The solution is a generalization of the method previously reported for selecting the code-group transcriber over the normal COMDEC transcription modules. CATNIP can now inform COMDEC that its understanding of the transmission indicates that a given selection of vocabularies would be appropriate for the next segment of the transmission. COMDEC accepts the selection and associates it with that point in the transmission for future reference. At the end of the segment, the "normal" vocabulary is restored if there is no new selection made by CATNIP. Note that the selection of vocabularies does not take effect immediately: it is deferred until the next logical break in the code (usually the occurrence of a "break" symbol). This method is used because the indication that English text will be transmitted usually occurs as part of a sequence of information describing the message about to be transmitted.

This mechanism could be further generalized to allow appropriate CATNIP diagrams to trigger any desired vocabulary change, for example, loading a special vocabulary appropriate to a particular sender-receiver pair.

Another change to COMDEC involved an attempt to preserve knowledge of some sender characteristics across transmissions by other senders (Lebling). Specifically, many chatter transmissions are too short for the initial MAUDE-like transcription module of COMDEC to get good values for the durations of code elements. COMDEC now preserves those values in a structure associated with the sender and uses it as a base from which to calculate the new values if that sender is encountered again. The ability to preserve this sender information makes a significant difference in the quality of the transcription.

Such information could be preserved in a "sender characteristics" data base between runs of COMDEC.

Finally, COMDEC's primary home has been moved to the MIT-XX machine from MIT-DM.

1.a. Transcription of Code (Cipher) Groups

A new module, CODEPARSE (Kaiser), was added to COMDEC during the past year. CODEPARSE is an "expert" on the transcription of Morse "code-groups". Code-groups differ from both English and Morse-network "chatter", which are also transcribed by COMDEC, in that there is no vocabulary for code-groups: almost any five-character sequence is legal and most such sequences are equally likely to occur.

CODEPARSE makes two passes over a file of mark and space durations. The first pass involves determining some of the information described below. During the second pass, CODEPARSE uses this information and a best-fit algorithm to transcribe marks and spaces into code-groups. CODEPARSE requires the following information:

1. the mark type and space type hypotheses produced by COMDEC's version of the MAUDE transcriber
   (These hypotheses may be disproved by a group with the wrong number of characters, a character with more than six marks, or a group containing both letters and digits.),

2. the length of code-groups in the message
   (This is usually five characters, but it may be longer or shorter. All code-groups in the same message have the same length. CODEPARSE determines this length by generating a histogram of group-length frequency on the groups produced by MAUDE.),

3. the number of code-groups in the message
   (This information is passed to CODEPARSE by CATNIP, the ATN parser, which extracts it from the message header. The length may be indeterminate.),

4. the expected end of the message
   (This is determined by COMDEC, which looks for the first "break" symbol (a Morse "pro-sign" spelled "BT") or speaker change following the message header. CODEPARSE halts after it has either produced the expected number of groups or reached the expected end of the message, whichever comes first.),

5. whether the message is composed of number groups or letter groups
   (Some combinations of marks may represent either letters or digits, depending on context, e.g. "dot dash" may mean "A" or may be a "cut" version of "1". CODEPARSE uses different marks-to-character dictionaries for these two cases. The module determines which dictionary to use by generating a histogram of the frequency of unambiguous mark combinations.), and

6. the knowledge that operators sometimes send error-signs, which are series of dots that indicate the presence of a recognized error
   (A set of heuristics is used to recognize error-signs and determine how much of the previous transmission was garbled and should be discarded.)

The program is at least as successful as a human operator at correcting errors in the samples of Morse code available to us. Below is an example where CODEPARSE performs significantly better than a human operator. The first set of code-groups is the original script used by the human sender, and it is followed by the MAUDE transcription. The transcription produced by CODEPARSE has six errors (denoted here by lower-case letters) and the transcription produced by a human operator has nine errors. Note that all except one of the errors in the CODEPARSE transcription involve a single character substitution, while most of the human operator's errors are horribly garbled. This supports the view that the sending operator made some mistakes in transmitting the message. It appears that CODEPARSE correctly transcribed all of what was actually sent except one code-group which the human operator transcribed correctly but CODEPARSE did not.

Script:

```
NAPOF OTHRQ AFQMU FTFWZ RFXLN EXFMY GZNPS KMBKH DBRJN EQHOB QTDMZ
XTZWU BNZUW JVUVT 53885 QISKO VTPPM VNJAG ZXHTK VXTKV SPSPK KJNRU
TWOXQ CWEXB OMDIJ FMMOM LXDUU IZVVF JCNAR UOPTV NJHDB XUQSS HFOAK
SUOPR
```

Maude:

```
N{.-.--.} OF O THRQAFQ{--..-} FTFWZRFXLN E{-..-..-.}MY G Z NPS KMBKH D 6
EEEEEEEEE D BRJN EQV EEEEEEE E QHOB QTDMZXTZ OU BNZ{..-.--} E OVUVT 3 885
QIS KOV{-.--.}PM VN{.---.-}G ZXH TTA {V-.-}TKV EIWE S WE K KJ N R U T W O X Q C
W E X B O M D I J F M M O M L X DUA IZ{...-..}TFJCNAR{..----}WETVNJHDBXUQSH
HFOAKSUOP R
```

CODEPARSE:

```
NAPOF OTHRQ afqtx FTFWZ RFXLN EXFMY GZNPS KMBKH DBRJN EQHOB QTDMZ xtzou
BNZUW JVUVT 53885 QISKO VTPPM VNJAG ZXHTK vktkv SPSPK KJNRU TWOXQ CWEXB
OMDIJ FMMOM lxdua izvuf JCNAR UOPTV NJHDB xuqsh HFOAK SUOPR
```

Human:

```
NAPOF OTHRQ AFQMU FTFWZ RFXLN EXFMY GZNPS KMBKH DBRJN EQHOB QTDMZ
XTZWU BNZUW edvuv (03885) QISKO VTPPM VNJAG zxhtq 3.k.v SPSPK KJNRU TWOXQ
CWEXB OMDIJ FMMOM LXDUU imivitf JCNAR uowet vnjhd bxuqsh HFOAK suojr
```

2. CATNIP Chatter and Header Understanding System

The final component of COMCO-1 is CATNIP (COMCO-1 Augmented Transition Network Interfaced Parser), a parser that uses an ATN grammar to choose a path through the lattice of possible transcriptions produced by COMDEC (Kaiser, Sherry, Phillips). During the past year, the major effort involving CATNIP has been a revision of the design of this grammar. The purpose of the revision was to devise a better means of representing the structure imposed on Morse-code conversations by radio-network protocol. The new design is described below. It is assumed that the reader is familiar with the concept of an augmented transition network grammar; a formal definition and discussion can be found in Woods' article [2].

The grammar is now organized into topical as well as syntactic categories. Although ATNs are usually organized into subnetworks that process syntactic structures--such as "noun phrase" in English--each of the twenty or so subnetworks that make up CATNIP's grammar is designed to process a particular set of semantically related phrases. This unusual organization results from the weakness of syntax for the chatter language; most structure can be described in more revealing terms as a result of semantic considerations.

Chatter has only two kinds of syntactic rules. The first kind is characterized by the following example: if either of the constructs "DE ___" ("This is station ___") or "___ DE ___" ("Station ___, this is station ___") occurs in a transmission, it occurs before contact has been established. These two constructs are different ways of identifying a new operator who is beginning transmission; either can occur in any position where self-identification of an operator is desired: logically this is at the beginning of a transmission by that operator. The syntactic rule is replaced by a more intuitive semantic rule that groups the two phrases in the topical category "Identification of Operators", denoted "ID-OP" in the grammar.

The second type of syntactic rule is the ordering of the "arguments" that follow almost all Morse-code Q-signs and many other words, e.g. "QSY 3500" ("Please change to transmission on frequency 3500 kilohertz"). The definition of the particular Q-sign includes a set of ordered informational "slots" that should be filled by the Q-sign's arguments (for example, "QSY" alone means "Change to transmission on frequency ___"). However, the properties of the argument words depend not only on the lexical features of the particular word of which they are arguments, but also on the context. For example, in the phrase "NR 1 GR 200 QTR 1500" ("message number 1, with 200 code-groups, at 1500 hours"), "GR" is followed by a single number. However, in phrases like "PSE RPT GR 10 , 20 , 30" ("Please repeat code-groups 10, 20, and 30"), the arguments of "GR" are one or more numbers separated by delimiters.

Although the syntax of chatter is weak, there is a strong discourse structure imposed by radio-network protocol. First, the operators involved must establish contact with each other; this is represented by the "CONTACT" subnetwork in the ATN. One of the operators prepares to send a message and then sends it, either as code-groups or English text; this process is modeled by the "TRAFFIC" subnetwork. The receiver may ask for several words to be repeated and eventually acknowledges receipt of the message. This process is modeled by the "RPT-INFO" subnetwork. "TRAFFIC" and "RPT-INFO" are repeated until all operators have sent all their prepared messages. Then the operators begin signing off, which may involve negotiations regarding re-establishment of contact at some future time: this is represented by the "END-CNCT" subnetwork. At this point, the conversation may terminate, or an operator may continue by trying to establish contact with a new operator.

The main topics of discourse are broken down into additional subnetworks based on topical categories. For example, "CONTACT" has transitions indicating use of the lower-level subnetworks "ID-OP" (identification of operators), "NET-RELAY" (the setup of a network relay, between two operators that cannot communicate directly, through a third operator that can communicate with both other operators), and "QUAL-CNCT" (discussion of signal characteristics). It is only within these lowest-level subnetworks that syntactic structure shows up, for example, in the ordering of Q-sign arguments; but, as discussed above, this structure is the result of semantic considerations. The only remaining syntactic rule is that arguments do follow the word of which they are arguments.

With one major exception, these four topics are the only possibilities for discussion and they always occur in this order. The exception is the "Interrupt Diagram" ("diagram" is the name of the subnetwork data type of CATNIP's grammar), denoted "INTRUPT" in CATNIP's grammar, which can be invoked from any state and represents an interruption in the smooth flow of transmission. The possible types of interruptions include a third operator suddenly breaking in on a conversation; static on the air waves, which must be dealt with by changes in transmitter characteristics or frequency; and so on. These interruptions are very difficult to parse--since the context is made invalid by the break--and present an interesting problem that has been only partially solved by the development of the Interrupt Diagram.

In addition to this reorganization of the transition-network component of the grammar, the set of registers and associated functions has also been revised significantly in order to deal more effectively with the information conveyed during Morse-code conversations. This information includes the names and locations of the operators involved; the status of their stations and transmitters, including ratings of strength, clarity, etc. of signals; the general status of the radio network; header information for each message (identification number, number of words, time of transmission); and each message body. The final contents of the registers containing this information constitutes the "result" of a parse.

It is also necessary to store contextual information, such as the last word transmitted, the number of code-groups received so far in the current message, and a list of pending questions and requests for each operator. The contents of these registers are useful for error-recovery and for determining the meaning of ambiguous transmissions like "5 K", which is probably the response to a question asked during the last transmission of another operator. Most of the associated functions simply select words and phrases from a transmission and put them in the appropriate registers, or delete information that is no longer needed (e.g. a question that has been answered).

Tests are generally associated with transitions that have generic symbols, e.g. "call-sign" (operator's call-sign or name), "#" (number), and "any" (English text or code-group), particularly the last. (By convention, a generic symbol is represented in lower case in CATNIP's grammar, and it is replaced by the appropriate chatter word at the time of parsing.) Additional conditions are necessary because "any" matches any word. The two tests associated with every occurrence of "any" in the grammar are (GROUP? input)--which is TRUE if the input token is not a Q-sign or delimiter--and (NOT? <list>)--which is TRUE if the input token is not a member of <list>, a list of chatter words that should not be accepted as code-groups or English words.

Besides this revision of the grammar, some work has also been done on the parser itself during the past year. The data types it uses have been reformatted, and many programs have been rewritten with the intention of decreasing processing time, which was a problem with the earlier version.

3. Grammar Extension

MAGE is a computer program (Kaiser) that performs grammatical extension for the augmented transition network grammar used by CATNIP to parse conversations over Morse-code radio networks. Grammatical extension is the process of receiving individual examples of "legal" sentences that are not already accepted by the grammar in question, and extending the grammar to include some mechanism so a parser using the grammar can parse the example, plus a large number of "similar" sentences.

The development of MAGE was motivated by a problem with CATNIP's ATN grammar: the grammar was inferred by hand from several hours of transcripts. Unfortunately, these transcripts do not represent the complete set of vocabulary words and phrase structures that are transmitted over real radio networks. (It is hoped, however, that they represent radio-network protocol fairly accurately.) Therefore, it was desirable to develop some mechanism that facilitates the extension of CATNIP's grammar to handle additional words and phrases. MAGE is an implementation of this mechanism.

MAGE is not capable of inferring a grammar from scratch; it builds on a "core" grammar, which includes a transition network, registers, a set of pre-coded functions for tests and actions, and a dictionary of the internationally defined Q-signs (but not other chatter words). The grammar is extended incrementally: each example is successfully "learned" before the next example is provided. The incremental feature is necessary in the Morse-code domain because the grammar can never be complete: chatter is an evolving, "spoken" language. The extension mechanism must always be ready to add one more phrase to the grammar.

MAGE involves three components: a world model derived from knowledge about the Morse-code domain and the ATN formalism; a procedure for constructing hypothetical extensions to the grammar; and an evaluation measure that selects from among these hypotheses one extension that provides the parser using the grammar with the ability to "understand" the particular example.

The world model includes everything MAGE knows about the domain: the core grammar, a table of synonyms for Q-signs and other chatter words that appear on transitions in the core grammar, and information about the ATN formalism and its implementation. In the situation of extending CATNIP's grammar, MAGE would use CATNIP's grammar as its "core", and transmissions excerpted from new transcripts would be provided as examples. This procedure would enable CATNIP to acquire the ability to parse Morse-code conversations that it may not have been able to handle previously. Additional transcripts are not currently available, however, so MAGE has been tested using a small subset of CATNIP's grammar as its core and other transmissions from the original transcripts as examples. These tests have shown that MAGE is capable of acquiring an ATN grammar equivalent to CATNIP's grammar, without the "Interrupt Diagram". MAGE is not yet able to recognize interrupt situations and extend a grammar accordingly.

The extension-construction procedure includes two algorithms. The first produces potential extensions to the transition-network (TN) component of the grammar. The evaluation measure selects the first one of these structural extensions that enables the grammar to accept the current example. The second phase enumerates a set of test/action specifications for the transitions in the chosen extension. The evaluation measure selects some subset of these that enables a parser using the grammar to "understand" the current example, in the sense of extracting the important information conveyed by the example and placing it in context.

The first phase requires partial parsing of the current example. At the point where the parser is left dangling by an unknown phrase, the first algorithm compares the location in the transition network where the parser hangs up, and the position in the network where the example would resume matching (the existence of such a position is guaranteed because "end of example" matches any terminal state by default), to the models presented below.

This set of models represents all single-transition extensions to the general three-state finite state machine (FSM) shown in Figure 1. (A single subnetwork, without tests and actions, is a FSM.) Model 0 (Figure 1) represents the original status of a subnetwork. The circle containing an "S" represents the state, or set of states, where the parser hung up. It is possible for this to be a set of states because of a modification of the parsing algorithm to consider multiple start-states and the nondeterministic nature of the ATN. The circle with the darkened area denotes a terminal state. The single intermediate state represents the arbitrarily intricate pattern of transitions and states between the start-state and one terminal state in an actual subnetwork.

Each of the models illustrated in Figures 2 through 8 represents a general one-transition extension to model 0. All extensions that are possible, considering the chatter domain, are included in this set. The new transition in each model represents an arbitrarily long string of transmission/next-state pairs. The first state transition in this string leaves a state in the original subnetwork as shown in the particular model; the final transition is connected to the next-state shown.

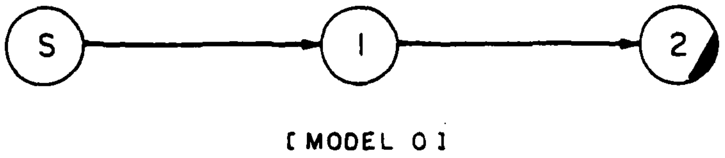


Figure 1. Model 0: General subnetwork.

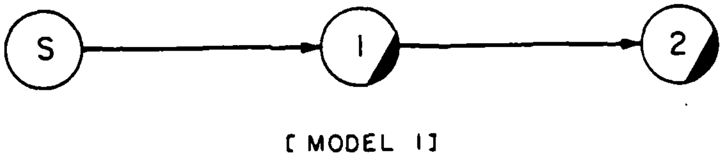


Figure 2. Model 1: Intermediate state becomes terminal state.

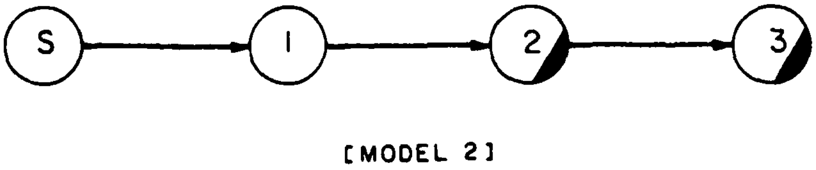
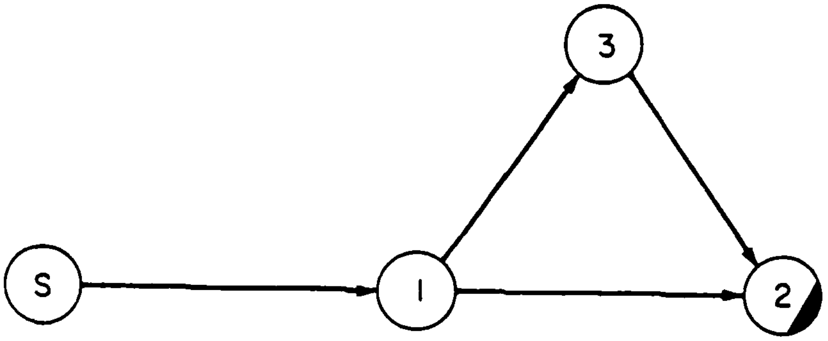
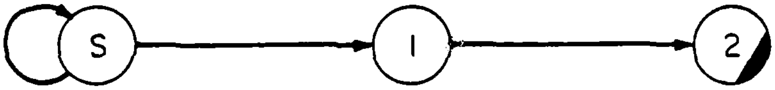


Figure 3. Model 2: Terminal state becomes intermediate state.



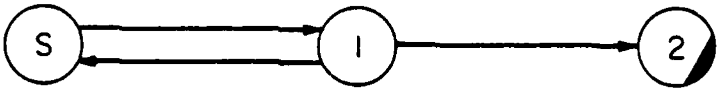
[MODEL 3]

Figure 4. Model 3: State(s) inserted between adjacent states.



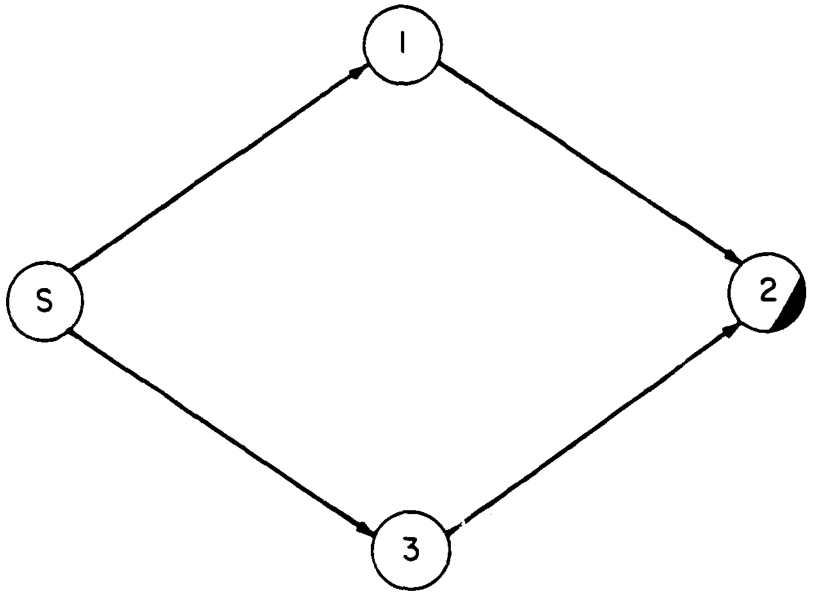
[MODEL 4]

Figure 5. Model 4: Transition loops to same state.



[MODEL 5]

Figure 6. Model 5: Transition returns to previously visited state.



[MODEL 6]

Figure 7. Model 6: Path inserted between start and terminal states.

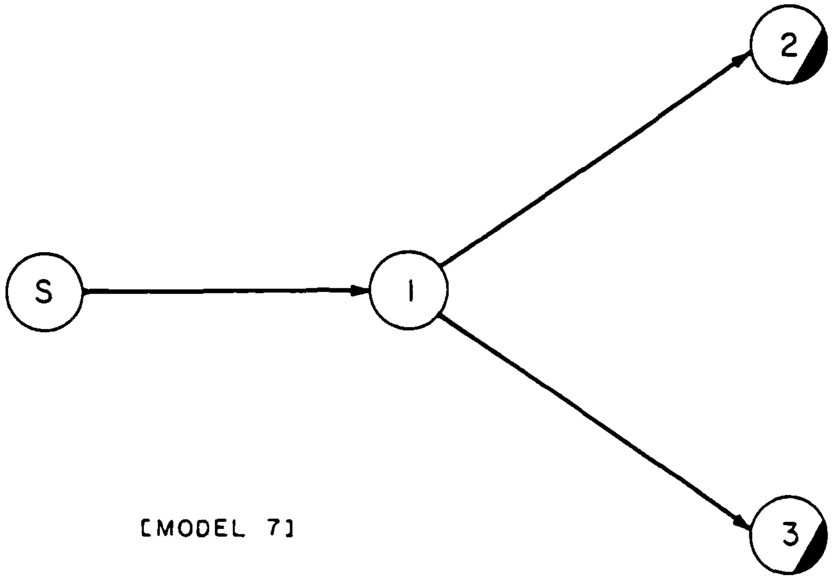


Figure 8. Model 7: New path ending in final state.

Every model/next-state pair produced by this algorithm is subjected to an evaluation that selects the first pair that passes its criterion: "Will this structural extension place the new phrase in the correct topical context?" This criterion is fairly simple to test for phrases containing at least one Q-sign, since Q-signs are associated a priori with the two or three subnetworks in which the Q-sign might make sense. The context of the rest of the example is generally sufficient to determine uniquely the appropriate topical category for the new phrase.

Those phrases that do not contain Q-signs are more difficult to evaluate. When an example contains an unknown word, MAGE asks the user if it is a synonym of any known word and, if so, which one. In general, one or more of the words in each example have known meanings, which can be used to determine the meaning, or at least topic, of any unknown words. The discourse topic is used to place the new phrase in the appropriate subnetwork. When the evaluation measure has selected a single structural extension, the test/action phase of the extension procedure takes over.

The enumeration algorithm for tests and actions is very simple. Each test and action specification is associated a priori with a set of chatter words. As new chatter words are "learned", these are also associated with tests and actions according to their presumed informational content. The algorithm simply looks at the symbol on each transition in the structural extension and returns the test/action specifications associated with that symbol. However, it is not likely that the entire set produced here should be associated with the new transition.

The evaluation criterion for augmenting a transition with a particular test or set of tests is this: "Will this test or set of tests ensure that all words accepted by this transition are meaningful in the current context?" An action or set of actions is approved for a transition if those actions will select and save the important information contained in this phrase, and ignore any meaningless words. When a set of tests and actions has been associated with each transition in the chosen extension, that extension is permanently added to the core grammar. The addition is "permanent" in the sense that it can now aid in the bootstrap process described above.

MAGE now operates independently of CATNIP, except that it uses a small subset of CATNIP's ATN grammar as its core grammar and is theoretically capable of extending the real CATNIP grammar when new transcripts become available. However, it may be possible to integrate the two programs. CATNIP could be modified to switch between parsing mode and extending mode (MAGE) whenever it receives some transmission that it is not able to parse. However, this may cause some timing conflicts since the ATN parsing algorithm implemented by CATNIP expects a static data base.

C. DATA-INTENSIVE PLANNING

This is a new area of research and development, begun in January 1979. Except for the communication aspect, the rest of this work is in an exploratory stage.

1. Interpersonal Communication

The research in interpersonal communication has continued at a low level with further design and implementation of the Data-based Message Service (DMS) [1] as described below (Broos, Galley, Vezza). DMS is "data-based" because the messages it manages are data in a number of similar on-line relational data bases, which may contain thousands or even tens or hundreds of thousands of messages.

The general model on which DMS is designed is that of a typical office. The interface at an intelligent terminal between DMS and a user is designed to be comfortable and familiar to people not used to working with computers, rather than employing computer terminology. Concepts and terminology from typical office methods of managing paper-based messages (letters, memos, and so on) are used wherever possible.

1.a. Terminal driver

The DMS message system [5] originally utilized an 8080-based Hewlett-Packard model 2649 display terminal with 40 kilobytes of firmware specially developed by USC-ISI. This intelligent terminal supports such features as multiple independently-rollable display "windows", selectively highlighted characters, local editing functions, "non-enterable" and "non-editable" areas, and demand "paging" of text from the mainframe. There are, in addition, some features which appear to be terminal functions, but which are actually performed by the virtual terminal (VT) module of DMS (Berez), which handles communication with the terminal, responds to its demand paging requests, and so on. The VT features include: user-set visible marks to delimit a text region to be copied, deleted, or used as a search key; user-set invisible "bookmarks" that can be jumped to; separation of each command's output so it can be brought back for later reference; and, in the military version of the VT, highlighted security-level tokens and function keys associated with text fields.

Based on our experiences with the DMS system, we wanted to be able to run that system and other similar systems without having to have the special terminals and their firmware. In other words, we wanted to replace the current VT with another module which would present essentially the same functionality, but would be able to utilize any moderately capable display terminal as its I/O device.

This new VT (Broos) is table-driven to a large extent, so that new types of terminals can be interfaced to it with a minimum of difficulty. It has tables to support Hewlett-Packard 2645, Imlac PDS-1D, DEC VT52 & VT100, and ITS standard "software" display terminals. It has a model of each terminal's capabilities and the special character sequences associated with its functions (such as "insert line") and status (such as user-modifiable strappings). DMS can query the VT -- to find out whether a text field has been modified or not -- and get a copy of or pointer to a modified field. DMS actually executed a standard script of commands slightly faster with this new VT, and the response time is subjectively shorter. Presumably this speed results from the smaller working set and more frequent or more uniformly-spaced accesses to various pages--because the new VT uses the same run-time support as DMS proper. In short, DMS can now be used to its fullest with this new VT and readily available terminals, rather than with the custom terminals previously required.

1.b. Other changes

The modules that manage indexes for the message data base were upgraded (Galley) to manage indexes of arbitrary size, by using a segmentation mechanism to split a tree-like index into branches of manageable size.

1.c. Reader and Comsys

A new message reader-composer program named "Reader" was written in the past year (Lebling). It was intended to be stop-gap effort to fill in until the DMS reader-composer is modified to deal with the local environment. It was also intended to explore some of the DMS ideas in a different environment with a different kind of users.

Reader incorporates, almost unmodified, the office model of DMS. It operates on messages produced by the "Comsys" message daemon, and produces messages for that daemon to transmit. Messages are maintained in a central store accessible to all users and are copied to the user's storage area once they reach a certain age.

The message composition part of Reader is the "POD" message composer that has been running on MIT-DM for several years. Since it is a MDL package, it was essentially trivial to incorporate it bodily into Reader. Due to the MDL library system, Reader users share this code with those using POD only as a composer. Reader also contains a significant amount of code from Comsys, which is shared with the daemon itself.

In addition to modifications to enhance interaction with Reader, Comsys underwent a major redesign of its data storage mechanisms in the past year. The major purpose was to speed up the daemon's operations, which in the previous implementation consumed a significant fraction of the system resources. A new data storage mechanism for Comsys was designed and implemented, using the MADMAN/ASYLUM object-oriented file system originally developed for DMS [1].

2. Data-Base Alerting

Work has begun on determining the state of the art in data base alerting mechanisms and in considering various issues in the implementation of alerting mechanisms suitable for use in data intensive planning systems (Noss). Most of the work has been in comparing current literature on related concepts (notably simple alerters and triggers for maintenance of semantic integrity) with the desired characteristics of a general alerting system.

The model alerting system envisioned is one which supports efficient mechanisms which cause a message to be sent to a human planner either when a specified event occurs in the data base system, or when a specified time has elapsed without the occurrence of a certain event. The installation and removal of alerters in the data base system should be similar in syntactic power and even form to a query. This would have several desirable results: alerters would be easy to install and remove; the predicate specifying the condition to be monitored by the alerter would be powerful enough to retrieve any information available in the normal query language of the system; and, in high-level relational languages, optimization of the alerter predicate would be simplified.

Efficient operation of alerters is of crucial importance in any real system. There may be arbitrarily many update operations for each alert, so each alerter predicate should be expressed in a way which minimizes the retrievals and computation necessary during an update. Just as the actual firing of an alerter is a comparatively rare event, even the complete evaluation of its predicate may rarely be necessary. In many cases of practical importance, it can be determined that no retrievals at all are necessary to show that the specified condition cannot be satisfied. For example, an alerter waiting for a certain total to exceed a certain value is not going to fire on a transaction which does not increase any of the constituent parts of the total. The procedures used when an update is done should be compiled in a way which eliminates such unnecessary tests. Recent research in the closely related area of semantic integrity triggers is helpful but not directly transferable, since such triggers may always legitimately make assumptions about the truth of their own predicates before a transaction takes place and since alerters need not act in real time before a transaction is complete, allowing optimizations such as batch testing of updates.

D. OTHER PROJECTS

1. MDL Development

With the acquisition of the MIT-XX DECsystem-20 computer, it became necessary to create a "liveable" MDL environment on its Tops-20 operating system. Previously, MDL itself had been modified to run on Tenex and Tops-20, but little attempt had been made to bring the full MDL environment up on the other operating systems.

As a major part of this effort, releases 55 and 105 (for ITS and Tops-20, respectively) of MDL were released (Blank, Lebling, Anderson, Reeve, Berkowitz).

There were two major deficiencies in previous Tops-20 versions of MDL. First was the lack of display-terminal support. MDL release 105 corrects this by including display support for specific terminal types known to the operating system (Imlacs, VT100s, etc.), pending more general display support in the Tops-20 operating system.

The second major deficiency in Tops-20 MDL was its primitive version of the MDL library system, and its incompatibility with the ITS MDL library system. The ITS version of MDL supported a central library of public software and a library of pure, sharable binary code. The Tops-20 version of the language supported only a central file-system directory for both public software and pure, sharable code. The ability to load modules dynamically did not exist.

It was decided to modify the library systems on both ITS and Tops-20, to make the user interfaces identical and to bring the Tops-20 implementation up to the level of the ITS implementation. At the same time certain minor problems with the ITS library implementation were corrected. As a consequence, MDL releases 55 and 105 have a library system that is independent of operating system and is significantly better than even the previous ITS implementation.

Other changes included in MDL release 105 were intended to further reduce the inconsistencies between the ITS and Tops-20 versions. Most differences remaining are due to incompatible differences in the two operating systems. Most modules of MDL code are directly transferable from ITS to Tops-20 without change. For modules that must interact with incompatible features, a mechanism has been developed for determining whether a given feature is present in the run-time environment.

Development of Tops-20 MDL and its environment is expected to continue, but most major subsystems (including the MDL compiler) now exist on both systems.

A document was produced whose intent was to be a reference manual for many of the programming aids which exist for the MDL user. The document [3] subsumes many older, obsolete documents and contains a large amount of new material. It supplements the bare-bones approach of "The MDL Programming Language" [4] by discussing the real conventions and mechanisms in use by MDL programmers. The latter manual was updated (Galley) to reflect the newest releases of the MDL interpreter. Parts were substantially re-written in an attempt to make them more understandable, and several interpreter bugs were discovered.

The cross-referencing listing generator (MAT), source-file comparer (MUDCOM), compiler user interface (COMBAT), and various other ITS programs essential to a congenial MDL environment were converted to run on Tops-20. In all cases, a common source with assembly-time conditionals for operating system dependencies was maintained (Blank, Anderson, Lebling).

2. Rmode

Rmode, a real-time display text editor, was modified to run on Tops-20 (Lebling), and now has a common source compatible with both ITS and Tops-20.

E. REFERENCES

1. MIT Laboratory for Computer Science. Progress Report XV. Cambridge, Ma., 1979.

2. Woods, William A. "Transition network grammars for natural language analysis," Communications ACM, 13, 10 (October 1970), 591-606.

3. Lebling, P. David. "The MDL programming environment," MIT, Laboratory for Computer Science, Cambridge, Ma, 1979.

4. Galley, S.W. and Pfister, G. "The MDL Programming Language," (revised and updated edition of "MDL primer and manual"), MIT, Laboratory for Computer Science, Cambridge, Ma., 1979.

Publications

1. Lebling, P. D., Blank, Marc S., and Anderson, Timothy A. "Zork: A computerized fantasy simulation game," IEEE Computer, April 1979.

2. Licklider, J.C.R. and Vezza, A. "Applications of information networks," Proceedings of the IEEE 66, 11 (November 1978), pp. 1330-1346.

3. Vezza, Albert, Lebling, P. David, et al, "Machine recognition and understanding of manual Morse," in Distributed Sensor Nets, Computer Science Dept., Carnegie-Mellon University, Pittsburgh, Pa., December 1978.

Theses Completed

1. Holt, D. "Computer aided task scheduling," S.B. thesis, MIT, Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., May 1979.

2. Kaiser, Gail E., "Automatic extension of an augmented transition network grammar for Morse code conversations," S.B. thesis, MIT, Dept. of Electrical Engineering and Computer Science, Cambridge, Ma., May 1979.

Talks

1. Licklider, J.C.R. "Problems and opportunities in government information systems," University of Maryland, April 19, 1979.

2. Licklider, J.C.R. "Information policy relating to the coherence of information systems," University of Michigan, March 22, 1979.

3. Licklider, J.C.R. "Education of students to cope with problems of computers and society," Computer Conference, Newcastle-upon-Tyne, England, September 5, 1978.

4. Licklider, J.C.R. "The view from here," Asilomar Conference Center, Asilomar, California, October 2, 1978.

5. Licklider, J.C.R. NATO Symposium, Crete, July 27-August 12, 1978.

6. Vezza, A. "Machine recognition and understanding of manual Morse," Distributed Sensor Nets Workshop IPTO/DARPA, Carnegie-Mellon University, Pittsburgh, Pa., December 8, 1978.

7. Vezza, A. "Overview of understanding Morse code," Defense Department Technical Analysts, Washington, D.C., October 5, 1978.

REAL TIME SYSTEMS GROUP

Academic Staff

S. A. Ward, Group Leader

M. L. Dertouzos

Research Staff

P. Houpt

Graduate Students

C. Baker
C. Cesar
J. Gula
R. Halstead
A. Mok
A. Reuveni

B. Schunck
J. Sieber
E. Strovink
T. Teixeira
C. Terman

Undergraduate Students

J. Arnold
D. Goddeau
T. Hayes
C. Law

R. McLellan
J. Mogul
S. Tomlinson

Support Staff

C. Eliot
E. Tervo

L. Zolman

A. INTRODUCTION

Research of the Real Time Systems group (until recently Domain Specific Systems Research group) is organized roughly around three major foci: 1) the Nu personal computer; 2) the MuNet multiprocessor architecture; and 3) CONSORT and related compiler technology. The latter two of these projects have been described in previous progress reports; the following paragraphs serve to illuminate their respective current status and goals. The Nu effort began in June, 1978.

B. THE LCS PERSONAL COMPUTER

Nu is a proposed standard computing resource for LCS (and eventually a larger community) for the next decade. It is intended to provide a path for the orderly evolution of computer usage patterns from the current dependence on large, centralized, multiuser systems to an eventual coherent network of single user ("personal") machines.

The design and construction of the initial Nu prototype, featuring an 8086 processor and 102-line raster-scan display, was undertaken in June 1978 by Rae McLellan, Chris Terman, and Steve Ward and completed over that summer. This prototype served to demonstrate feasibility and to attract industrial sponsors--particularly potential manufacturers--to the effort. It was instrumental in obtaining Exxon funding of research by Prof. Hammer's group in Office Automation, and has led to an agreement with the Heath Company for the manufacture of Nu.

In autumn 1978 an extensive redesign of Nu was initiated (by Dave Goddeau, McLellan, Terman, Ward) pursuant to its manufacture. The revised design is based on a bus architecture whose parameters and protocols are sufficiently general to survive several generations of microprocessor architecture, and which supports hardware configurations ranging from sophisticated terminals to high-performance multiprocessor/multibus systems. The first components of the revised Nu design have recently become operational at the laboratory. These include a Z8000 processor card (with a simple 1024-page memory map), ROM/Arbiter card, RAM card, and the NuBus backplane. Ten initial prototypes (including video and secondary storage) are expected to be constructed by Heath and delivered to LCS in the fall of 1979.

A revised design of the video control logic and refresh memory has been completed by Goddeau and is under construction. The revisions will support a variety of display formats, and eventually multiplane graphics (e.g., for color and grayscale display).

A high-performance microcodable dual-bus processor (called Rho) is under development by McLellan. This device may be configured to provide optional high-speed graphics, to serve as a gateway for the construction of multibus systems, or to support particular languages and applications. A prototype Rho processor is expected to be operational during the summer of 1979.

An extensible base-level operating system for Nu, called TRIX, is being implemented by Jon Sieber and is expected to be operational in an initial form during the summer of

1979. The initial implementation (in the C language) runs on a PDP11/40 and will be bootstrapped to Nu when an appropriate configuration (i.e., one with secondary storage) becomes available. TRIX implementations for other machines (e.g. the VAX) are being seriously contemplated.

C. CONSORT: COMPILE-TIME TECHNOLOGY

Continuing work on CONSORT (described in previous reports) by Al Mok, John Pershing, Tom Teixeira, and Ward has resulted in a completed prototype implementation and a brief filmed demonstration. The current implementation translates a source program consisting of a block diagram (which may be input graphically) and produces an object program which runs on a single 8080 microprocessor. Static control structures are devised by CONSORT to meet real time performance criteria specified (in the source diagram) as latency constraints which dictate minimum rates at which data values must propagate thru the diagram. A source diagram may consist of multiple pages, each corresponding, for example, to a particular control strategy; object-time linkage mechanism provides for orderly transition between pages (passing state information to maintain continuity) as the target system passes from one phase of its operation to the next.

Experience with the current CONSORT implementation has been both encouraging and suggestive. Although many aspects of this initial effort are tentative and unpolished, the general approach it illustrates seems well suited as a basis for the construction of powerful engineering tools. In addition, the simple, very high level semantics of CONSORT programs together with specification of concrete performance criteria provide a nearly ideal context for the development of a variety of radical program transformation and optimization techniques.

Current work on code generation systems by Terman and on radical optimizations by Teixeira are expected to provide the basis for a substantially more ambitious CONSORT reimplementation. Relevant scheduling and partitioning problems are currently under study by Mok.

D. MuNet: OBJECT-TIME TECHNOLOGY

Previously reported work by Halstead and Ward has led to the implementation, during the past year, of a prototype multiprocessor MuNet implementation based on LSI11s. While our limited experience with this system is hardly definitive it is encouraging, and is reported in the forthcoming Ph.D. thesis of Bert Halstead.

An implementation of the MuNet-specific programming language, MuSpeak, was completed by Eric Strovink and is described in his recent S. M. thesis. MuSpeak is a low-level, general-purpose systems language whose syntax and semantics draw heavily from those of C, and the current implementation cross-compiles from the 11/70 UNIX system. While the compiler allows programmer access to the underlying message-passing semantics of the MuNet, it translates a variety of conventional constructs to message-passing form, creating continuation objects and partitioning code between them as necessary.

Design of a MuNet-based operating system by Gula, described in his recent thesis, integrates object-oriented I/O, process and exception handling with the message-passing structure of the system.

Continuing work by Clark Baker is directed toward failure recovery and other reliability measures applicable to the MuNet. Continuing research by Asher Reuveni explores alternative event-oriented language semantics based on broadcast rather than message-passing communications.

Publications

1. Dertouzos, M.L. and Moses, J. Editors, The Computer Age: A Twenty Year View , MIT Press, Cambridge, Ma., 1979.

2. Houpt, P., Johnson, T., and Ward, S. "Control designed software which generates, downloads, and executes real-time microprocessor code," Sixteenth Annual Allerton Conference on Communication, Control, and Computing, Allerton, Ill., Oct. 1978.

3. Gula, J. "Operating system considerations for multiprocessor architectures," Proceedings Seventh Texas Conference on Computing Systems, Houston, Tx., October 1978, 7-1 - 7-6.

4. Halstead, R. "Object management on multiprocessor systems," Proceedings Seventh Texas Conference on Computing Systems, Houston, Tx., October 1978, 7-7 - 7-14.

5. Mok, A. and Dertouzos, M. "Multiprocessor scheduling in a hard real-time environment," Proceedings Seventh Texas Conference on Computing Systems, Houston, Tx., October 1978, 5-1 - 5-12, to appear in IEEE Transactions on Software Engineering.

6. Strovink, E. "Compilation strategies for multiprocessor message-passing systems," Proceedings Seventh Texas Conference on Computing Systems, Houston, Tx., October 1978, 7-15 - 7-20.

7. Teixeira, T. "Static priority interrupt scheduling," Proceedings Seventh Texas Conference on Computing Systems, Houston, Tx., October 1978, 5-13 - 5-18.

8. Terman, C. "Compiling programs to meet real time constraints," Proceedings Seventh Texas Conference on Computing Systems, Houston, Tx., October 1978, 5-19 - 5-25.

9. Ward, S. "Real-time plotting of approximate contour maps," Communications ACM 21, 9 (September 1978), 788-790.

10. Ward, S. and Halstead, R. "A syntactic theory of message passing," to appear in The Journal of the ACM.

11. Ward, S. "An approach to real-time computation," Proceedings Seventh Texas Conference on Computing Systems, Houston, Tx., October 1978, 5-26 - 5-34, to appear in IEEE Transactions on Software Engineering.

12. Ward, S. "The MuNet: A multiprocessor message-passing system architecture," Proceedings Seventh Texas Conference on Computing Systems, Houston, Tx., October 1978, 7-21 - 7-24.

Talks

1. Dertouzos, M.L. "Expected developments in the computer field," Invited lecture before Committee on Science and Public Policy, National Academy of Sciences, Berkeley, Ca., February 4, 1979.

2. Dertouzos, M.L. "Forefront research in computer science," Invited lecture, Royal Swedish Academy of Engineering Sciences, June 18, 1979.

3. Dertouzos, M.L. "Speculations on computing by the 2000," Keynote address, Seventh Texas Conference on Computing Systems, Houston, Tx., October 31, 1978.

4. Dertouzos, M.L. "Computers and information processing: A twenty year view," Keynote address, International Conference on Information Sciences and Systems, Patras, Greece, July 9-13, 1979.

5. Dertouzos, M.L. "Societal prospects and problems of information processing," Invited lecture, International Symposium on Computer, Man and Society, Haifa, Israel, October 23, 1979.

6. Dertouzos, M.L. "Computers for command and control," Invited Lecture, Technology Trends Colloquium, Department of Defense, Annapolis, Md., March 29-31, 1978.

7. Dertouzos, M.L. "The computer field," Invited lecture at *Science Writer's* Club, Boston, Ma., January 26, 1978.

8. Dertouzos, M.L. Panelist, U.S. Department of Defense Colloquium on Command and Control, MIT Endicott House, Dedham, Ma., March 13-14, 1979.

9. Dertouzos, M.L. Chairman and moderator, Computers and Society, Massachusetts Institute of Technology, Cambridge, Ma., April 8, 1979.

10. Halstead, R. "Reference tree networks,"
    Dept. of Computer Science, U.C. Berkeley, Berkeley, Ca., January 31, 1979.
    E.E. Dept., Carnegie-Mellon University, Pittsburgh, Pa., March 15, 1979.
    Bell Telephone Laboratories, Murray Hill, NJ, March 29, 1979.
    Computer Science Dept., Carnegie-Mellon University, Pittsburgh, Pa., Apr. 19, 1979.

11. "Prime Concern," 30 minute CBS telecast, October 11, 1978, 8:30 pm, "Personal Computing" with M. L. Dertouzos.

12. Terman, C. "Machine-independent code generation," G.E. Research and Development Laboratory, Schenectady, NY., May 10, 1979.

13. Ward, S. "The MuNet: A scalable multiprocessor architecture,"
    Xerox PARC, Palo Alto, Ca., January 1979.
    Stanford University (videotaped for limited distribution), Stanford, Ca., January 1979.

U.C. Berkeley, Systems Seminar, Berkeley, Ca., January 1979.

14. Ward, S. "The MuNet: An overview," Workshop on Real Time Symbolic Computation, Boston, Ma., November 1978.

15. Ward, S. "Scalable multiprocessor architectures," University of Rochester, Rochester, NY., April 1979.

16. Ward, S., "Nu: An approach to distributed computing," Fifth IEEE Workshop on Microprocessors, Asilomar, Ca., May 1979.

Theses Completed

1. Goddeau, D. "A multiple bit plane video controller," S.B. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., May 1979.

2. Gula, J. "A distributed operating system for an object based network," S.M. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., May 1979.

3. Halstead, R. "Reference tree networks," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., June 1979.

4. Law, C. "Optimizing programs in a block-structured computer architecture," S.B. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., May 1979.

5. McLellan, H. "A microprogrammable dual-bus processor," S.B. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., May 1979.

6. Mogul, J. "Attaching the VAX-11/780 to the local network," S.B. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., May 1979.

7. Strovink, E. "Compilation strategies for a multiple-processor message-passing system," S.M. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., May 1979.

8. Tomlinson, M. "The renaissance machine," S.B. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., May 1979.

Theses In Progress

1. Baker, C. "Reliable distributed object management schemes," S.M. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected August 1979.

2. Cesar, C. "Real-time emulation of hardware," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected August 1979.

3. Mok, A. K. "Multiprocessor system design with hard real-time constraints," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected June 1980.

4. Terman, C. J. "Compiling programs for a real-time environment," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected January 1980 (formerly titled "A code generator for real time programs").

5. Teixeira, T. J. "Radical optimizations in real-time programs," S.M. thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected January 1980.

6. Reuveni, A. "Investigation of implementation strategies for the event-based language EBL on distributed systems," Ph.D. dissertation, MIT, Department of Electrical Engineering and Computer Science, Cambridge, Ma., expected January 1980.

TECHNICAL SERVICES GROUP

Research Staff

K. T. Pogran, Group Leader

Undergraduate Students

P. Dinnerstein
S. J. Lee
G. Simpson

Support Staff

P. Baskin
R. G. Bisbee
O. Feingold
J. D. Ricchio

TECHNICAL SERVICES

A. INTRODUCTION

The Technical Services Group functions in both a research role and a support role within the Laboratory. In its research role, the group is responsible for the development and implementation of hardware for the LCS Local Network, which is part of the Laboratory's Distributed Systems effort. In its support role, the group provides a number of services to the Laboratory, including:

1. Liaison for the Laboratory's ARPANET IMP and TIP;
2. Maintenance of some of the Laboratory's equipment; in particular, maintenance of the Laboratory's collection of nearly one hundred Digital Equipment Corporation VT-52 CRT terminals;
3. Development and maintenance of an electronics laboratory facility for use by all LCS groups;
4. Management and maintenance of the LCS Network.

B. LCS NETWORK HARDWARE

1. The Version 1 Local Network Interface

1.a. Logic Debugging

Work in the first half of the year centered on completing the final debugging of the Local Network Interface, described in last year's report. This device, originally designed by a group at the University of California at Irvine, is the basic building block of the LCS Local Network, a 1 Mb/s token-control ring network. A good deal of time and effort was spent during the 1977-78 reporting year debugging this 350-IC device. During the first half of this year, we found and fixed the remaining logic bugs in the device. Some were simply logic design errors; others, more subtle flaws in timing, etc., became obvious only when two or more LNI's were connected in a ring.

Two LNI's had been fabricated and were being used for debugging by the end of the last reporting year. In early September, a third LNI was fabricated, incorporating the bug fixes made to the first two, and several other changes. This LNI became the "prime" debugging LNI. A fourth LNI was fabricated in December, and two more were assembled in January.

1.b. Transmission Subsystem

In early January, experiments were performed with the transmission subsystem of the LNI to determine the optimum configuration of drivers, transmission lines, current-regulating resistors, and opto-coupler receivers. This was done because the transmission subsystem, as originally designed by the group at UC-Irvine, would not operate over more than one hundred feet of shielded twisted pair. The experiments performed in early January resulted in a design that used the same drivers and receivers and that operated, error-free, over a 300-500 foot cable. While this was far below the advertised range of two thousand feet, it was adequate for our immediate needs, and we chose to use the improved circuit as it was, rather than embark upon a complete redesign of the transmission subsystem with different components.

1.c. Initial Ring Network for Hardware Test and Evaluation

In late January, we set up a three-node ring, for hardware test and evaluation purposes, comprising our PDP-11/40 "gateway" machine, and the PDP-11/70 UNIX and VAX-11/780 operated by the DSSR Group.

1.d. The UCLA Ring Network

We were asked by DARPA to provide two LNI's to a group at UCLA under Professor Gerald Popek, who, initially, was to Vhave been the recipient of the original LNI's as developed by the UC-Irvine group. To provide LNI's to UCLA in the most expedient way, we stripped all wires from the two original LNI's, which had undergone many, many wire changes, were difficult to work with and not quite up-to-date, and sent the chassis to our fabrication shop to be re-wired. This was accomplished in less time and at much less expense than the manufacture of two completely new LNI's. The installation at UCLA went smoothly; in one day we unpacked, set up, and checked out two LNI's.

1.e. LNI Fabrication

We expect that there will be a total of twelve Version 1 Local Network Interfaces, eight within the Laboratory and four at UCLA. All twelve LNI's should be operational by the beginning of August. In general, LNI fabrication has gone rather well. Most of the LNI's were wired to a somewhat outdated wire list, requiring about two hours of manual wiring changes before checkout could begin. The total effort required to complete the assembly and checkout of an LNI has averaged two and a half man-days--quite reasonable for a device the complexity of the LNI.

1.f. The Prototype Ring Network

Network software, such as LNI device drivers, Transmission Control Protocol software, and applications-level protocol software, has matured along with the LNI hardware. This effort is described in the report of the Computer Systems Research Group. With the availability of software, the hardware test network described previously evolved into an operational prototype network to serve the Laboratory. This network is expected to provide service to five host computer systems within the Laboratory, and have a total of seven nodes.

The two network nodes not considered "hosts" in the usual sense of the word are LSI-11 systems. One is to serve as the gateway between the LCS Network and the ARPANET, while the other is a Terminal Interface Unit, or TIU, which provides access to the network for four terminals.

The host systems that are to be part of the prototype network are:

System	Group
PDP-11/70 UNIX	Domain Specific Systems Research
PDP-11/40 UNIX	Computer Systems Research
PDP-11/20 stand-alone	Technical Services
VAX-11/780 VMS	Domain Specific Systems Research
DECSystem-20	Programming Technology

2. The Version 2 Local Network Interface

2.a. Motivation

The Version 1 LNI provides a 1 Mb/s data transmission facility. Although it works, and works reliably, it has a number of drawbacks:

1. It is a "first cut" design and is quite bulky and, hence, expensive.

2. It connects only to the PDP-11 Unibus.

3. Its transmission subsystem will not operate over more than five hundred feet of cable.

As part of the Laboratory's research effort in Distributed Systems, the Laboratory is procuring a large number of personal computers. These machines need to be linked together in a network that spans the entire Laboratory and that will have between one hundred and two hundred nodes. This need, coupled with the drawbacks of the Version 1 LNI, has given rise to a project to design and produce a much improved Version 2 LNI. This effort has been undertaken in conjunction with Proteon Associates, Inc., of Waltham, Massachusetts.

2.b. Goals

Our discussions with Proteon began in December. By early spring we had evolved some specific goals for the project. We would build a ring network with the following characteristics:

1. Data rate of 8-10 Mb/s.

2. Inter-node distance of 300-500 feet.

3. Network of 100-200 nodes.

4. Modular design, with host interfaces for:

   - PDP-11 Unibus
   - "Nu" Terminal
   - Others, as the need arises

It should be emphasized that these characteristics were design goals, not requirements. We would, for example, be willing to accept a lower data rate if the inter-node distance goal could not be met at the higher data rate. Of paramount importance was the goal that a network based on the Version 2 Local Network Interface could be installed at the same time as the LCS Nu Terminals were arriving, which was expected to be the fall of 1979. One major reason for working with a company such as Proteon was that they were willing to undertake the manufacture, as well as the design, of the Version 2 LNI.

2.c. Progress

Detailed design of the Version 2 LNI began in May. The logic structure of the LNI Input and Output machines is patterned after that of the Version 1 LNI, improved and simplified. Experiments conducted in June helped determine the transmission circuitry and transmission medium to be used, and established the capabilities of the selected combination. It is expected that the first prototypes of the Version 2 LNI, with PDP-11 Unibus interfaces, will be available for evaluation and use in September.

We have also evolved a plan for the physical arrangement of the network transmission medium. This scheme, which we term the "star-shaped ring", eases the problems of installation, maintenance, and management of a large-scale ring network. This work was reported in a paper by Jerome H. Saltzer and Kenneth T. Pogran, entitled "A Star-Shaped Ring Network with High Maintainability", presented at the NBS-Mitre Local Area Communications Network Symposium in Boston on May 7-9, 1979.

2.d. Implications for the Future

MIT has a growing need for a campus-wide data communications network. We believe that our experience in developing and implementing a network of over one hundred nodes will be applicable to the problem of devising a network of several thousand nodes, spanning a wider geographic area. To this end, we are endeavoring, as much as possible, to be aware of and to deal with the organizational issues associated with such a large-scale network.

3. Additional Hardware

We have had, as a goal, the development of a single-board ARPANET ("1822") interface for the PDP-11. Such an interface could be manufactured at far less cost than the interface currently offered by Digital Equipment Corporation; and could thus be more readily incorporated vvinto a "low-end" PDP-11 or LSI-11 ARPANET Gateway for the LCS Network.

Our first effort, conducted during the 1977-78 reporting year, with several undergraduate students, yielded an unreliable, non-reproducible interface that was difficult to debug further. This year, a new project was initiated by a full-time staff member, and we have now attained our goal. Our PDP-11 IMP interface occupies one PDP-11 "hex" board, and contains a full-duplex PDP-11 Direct Memory Access (DMA) Interface in addition to the IMP interface logic. The DMA Interface designed for this project will also be used for the Version 2 LNI PDP-11 interface.

C. SUPPORT ACTIVITIES

As mentioned in the introduction, the Technical Services group performs a number of support functions for the Laboratory. The most prominent of these activities are reported here.

1. ARPANET IMP and TIP Liaison

Oded Feingold serves as Liaison for the Laboratory's IMP and TIP. The MIT-TIP provides terminal access to the ARPANET for DARPA-sponsored network users within the Laboratory on the MIT campus, and throughout the Boston area. Four types of service are provided:

1. Hard-wired terminals within the Laboratory.

2. Dial-up MIT CENTREX lines for MIT users.

3. Dial-up outside lines for Boston area users.

4. Dial-up and leased lines provided by DARPA for specific outside users.

The first two types of lines serve users at LCS; by managing the last two types of lines for DARPA, we are fulfilling our quid-pro-quo for having the services of the TIP available to us.

The Liaison function includes providing information to TIP users, ensuring that the lines are in working order, and--of growing importance to DARPA and the Defense Communication Agency, the manager of the ARPANET--keeping records of authorized users to whom the outside TIP telephone number is given.

2. Terminal Maintenance

Joseph Ricchio is responsible for maintaining the Laboratory's Digital Equipment Corporations VT-52 CRT terminals. This is a sizeable task, as the Laboratory owns nearly one hundred VT-52's, and they have not exhibited great reliability. We switched to self-maintenance of the VT-52's for several reasons:

1. Cost. Most repair operations are board-swaps. Digital's minimum service call charge generally required paying for far more time than was actually spent by the field engineer in effecting repair of the terminal.

2. Responsiveness. In general, we can respond more quickly to a terminal repair call than Digital can. In addition, we can easily make minor adjustments or repairs for which it would not be worth the expense of calling in Digital field service.

3. Continuity. We keep a repair history of each terminal, and can note and deal with repeated failures. We can also monitor "flakey" terminals.

Our experience has not been a completely positive one, for we have found it difficult to maintain a sufficient stock of spare circuit boards. Digital's "Module Mailer" board repair service has been slow to repair boards we have sent them, and all too often, boards received have not been properly repaired. This has sometimes prevented us from repairing a terminal and returning it to its user as rapidly as we would like.

Publications

1. Clark, D., Pogran, K., and Reed, D. "An introduction to local area networks," Proceedings of the IEEE 66, 11 (November 1978), 1497-1517.

2. Saltzer, J. and Pogran, K. "A star-shaped ring network with high maintainability," Proceedings NBS-Mitre Local Area Communications Network Symposium, Boston, Ma., May 1979.

Talks

1. Pogran, K. and Clark, D. "Local networks," GTE-Sylvania Electronic Systems Group, Needham, Ma., December 1978.

2. Pogran, K. "An introduction to local area networks," Bell Laboratories, Holmdel, N.J., February 1979.

LCS PUBLICATIONS

TECHNICAL MEMORANDA

TM-10 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
(Formerly Programming Linguistics
Group Memo No. 6, June 12, 1970)
September 1970
AD 712-068

*TM-13 Graham, Robert M.
Use of High Level Languages
for Systems Programming
(Formerly Programming Linguistics
Group Memo No. 2, November 20, 1969)
September 1970
AD 711-965

*TM-14 Vogt, Carla M.
Suspension of Processes in a Multiprocessing Computer System
(Based on M.S. Thesis, EE Dept.,
February 1970)
September 1970
AD 713-989

*TM-15 Zilles, Stephen N.
An Expansion of the Data Structuring
Capabilities of PAL
(Based on M.S. Thesis, EE Dept.,
June 1970)
October 1970
AD 720-761

TMs 1-9 were never issued.

*TM-16 Bruere-Dawson, Gerard
Pseudo-Random Sequences
(Based on M.S. Thesis, EE Dept.,
June 1970)
October 1970
AD 713-852

*TM-17 Goodman, Leonard I.
Complexity Measures for Programming
Languages (Based on M.S. Thesis, EE Dept.,
September 1971)
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

*TM-20 Jones, Thomas L.
A Computer Model of Simple Forms
of Learning (Based on Ph.D. Thesis,
EE Dept., September 1970)
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
(Based on B.S. Thesis, ME Dept., May 1972)
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 Complexity Classes 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
B.S. 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
M.S. & E.E. Theses, 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
B.S. 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
M.S. 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
B.S., M.S. & E.E. Theses, EE Dept.
June 1975

*TM-67 Cardoza, Edward E.
Computational Complexity of the Word Problem for Commutative Semigroups
M.S. Thesis, EE & CS Dept.
October 1975

*TM-68 Weng, Kung-Song
Stream-Oriented Computation in Recursive Data Flow Schemas
M.S. Thesis, EE & CS Dept.
October 1975

*TM-69 Bayer, Paul J.
Improved Bounds on the Costs of Optimal and Balanced Binary Search Trees
M.S. 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
M.S. 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
M.S. Thesis, EE & CS Dept.
September 1976
AD A030-402

*TM-78 Benjamin, Arthur J.
Improving Information Storage Reliability
Using a Data Network
M.S. 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
B.S. 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
(formerly On 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 1977
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
M.S. Thesis, EE & CS Dept.
July 1977

TM-89 Rodriguez, Humberto Jr.
Measuring User Characteristics on the Multics System
B.S. Thesis, EE & CS Dept.
August 1977

*TM-90 d'Oliveira, Cecilia R.
An Analysis of Computer Decentralization
B.S. 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
M.S. Thesis, EE & CS Dept.
December 1977

*TM-94 Berez, Joel M.
A Dynamic Debugging System for MDL
B.S. 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
M.S. Thesis, EE & CS Dept.
February 1978

TM-100 Misunas, David P.
A Computer Architecture for Data-Flow Computation
M.S. 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

TM-110 Jaffe, Jeffrey M.
An Analysis of Preemptive Multiprocessor
Job Scheduling
September 1978

TM-111 Jaffe, Jeffrey M.
Bounds on the Scheduling of Typed Task Systems
September 1978

*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
M.S. 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
M.S. & E.E. Theses, 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
B.S. & M.S. Theses, 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.
Timstamps and Capability-Based Protection
in a Distributed Computer Facility
B.S. & M.S. Theses, 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

TECHNICAL REPORTS

*TR-1 Bobrow, Daniel G.
Natural Language Input for a Computer Problem Solving System,
Ph.D. Thesis, Math. Dept.
September 1964
AD 604-730

*TR-2 Raphael, Bertram
SIR: A Computer Program for Semantic Information Retrieval,
Ph.D. Thesis, 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

TRs 5, 9, 10, 15 were never issued

*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

*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. Thesis, EE Dept.
June 1965
AD 470-715

*TR-19 Russo, Francis John
A Heuristic Approach to Alternate Routing in a Job Shop,
B.S. & M.S. Theses, Sloan School
June 1965
AD 474-018

*TR-20 Wantman, Mayer Elihu
CALCULAID: An On-Line System for
Algebraic Computation and Analysis,
M.S. Thesis, Sloan School
September 1965
AD 474-019

*TR-21 Denning, Peter James
Queueing Models for File Memory Operation,
M.S. 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 Multiprogrammed 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,
M.S. Thesis, EE Dept.
March 1966
AD 631-396

*TR-26 Cheek, Thomas Burrell
Design of a Low-Cost Character Generator for Remote Computer Displays,
M.S. Thesis, EE Dept.
March 1966
AD 631-269

*TR-27 Edwards, Daniel James
OCAS - On-Line Cryptanalytic Aid System,
M.S. Thesis, EE Dept.
May 1966
AD 633-678

*TR-28 Smith, Arthur Anshel
Input/Output in Time-Shared, Segmented,
Multiprocessor Systems,
M.S. Thesis, EE Dept.
June 1966
AD 637-215

*TR-29 Ivie, Evan Leon
Search Procedures Based on Measures
of Relatedness between Documents,
Ph.D. Thesis, 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,
M.S. Thesis, EE Dept.
August 1966
AD 637-192

*TR-32 Teitelman, Warren
PILOT: A Step Toward Man-Computer
Symbiosis,
Ph.D. Thesis, Math. Dept.
September 1966
AD 638-446

*TR-33 Norton, Lewis M.
ADEPT - A Heuristic Program for
Proving Theorems of Group Theory,
Ph.D. Thesis, Math. Dept.
October 1966
AD 645-660

*TR-34 Van Horn, Earl C., Jr.
Computer Design for Asynchronously
Reproducible Multiprocessing,
Ph.D. Thesis, EE Dept.
November 1966
AD 650-407

*TR-35 Fenichel, Robert R.
An On-Line System for Algebraic Manipulation,
Ph.D. Thesis, Appl. Math. (Harvard)
December 1966
AD 657-282

*TR-36 Martin, William A.
Symbolic Mathematical Laboratory,
Ph.D. Thesis, EE Dept.
January 1967
AD 657-283

*TR-37 Guzman-Arenas, Adolfo
Some Aspects of Pattern Recognition by Computer,
M.S. 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. Thesis, EE Dept.
August 1967
AD 662-224

*TR-44 Gorry, G. Anthony
A System for Computer-Aided Diagnosis,
Ph.D. Thesis, Sloan School
September 1967
AD 662-665

*TR-45 Leal-Cantu, Nestor
On the Simulation of Dynamic Systems
with Lumped Parameters and Time Delays,
M.S. Thesis, ME Dept.
October 1967
AD 663-502

*TR-46 Alsop, Joseph W.
A Canonic Translator,
B.S. Thesis, EE Dept.
November 1967
AD 663-503

*TR-47 Moses, Joel
Symbolic Integration,
Ph.D. Thesis, Math. Dept.
December 1967
AD 662-666

*TR-48 Jones, Malcolm M.
Incremental Simulation on a Time-Shared Computer,
Ph.D. Thesis, Sloan School
January 1968
AD 662-225

*TR-49 Luconi, Fred L.
Asynchronous Computational Structures,
Ph.D Thesis, EE Dept.
February 1968
AD 667-602

*TR-50 Denning, Peter J.
Resource Allocation in Multiprocess
Computer Systems,
Ph.D. Thesis, EE Dept.
May 1968
AD 675-554

*TR-51 Charniak, Eugene
CARPS, A Program which Solves
Calculus Word Problems,
M.S. Thesis, EE Dept.
July 1968
AD 673-670

TR-52 Deitel, Harvey M.
Absentee Comoutations in a Multiple-Access
Computer System,
M.S. Thesis, EE Dept.
August 1968
AD 684-738

*TR-53 Slutz, Donald R.
The Flow Graph Schemata Model of
Parallel Computation,
Ph.D. Thesis, 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,
M.S. Thesis, EE Dept.
October 1968
AD 689-468

*TR-55 Rappaport, Robert L.
Implementing Multi-Process Primitives
in a Multiplexed Computer System,
M.S. Thesis, EE Dept.
November 1968
AD 689-469

*TR-56 Thornhill, Daniel E., Robert H. Stotz, Douglas T. Ross
and John E. Ward (ESL-R-356)
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. Thesis, 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,
M.S. Thesis, EE Dept.
January 1969
AD 686-988

*TR-59 Guzman, Adolfo
Computer Recognition of Three-Dimensional Objects in a Visual Scene,
Ph.D. Thesis, 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. Thesis, EE Dept.
April 1969
AD 689-305

*TR-61 Baecker, Ronald M.
Interactive Computer-Mediated Animation,
Ph.D. Thesis, EE Dept.
June 1969
AD 690-887

*TR-62 Tillman, Coyt C., Jr. (ESL-R-395)
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. (ESL-R-398)
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. Thesis, EE Dept.
October 1969
AD 699-501

*TR-66 Beyer, Wendell T.
Recognition of Topological Invariants by Iterative Arrays,
Ph.D. Thesis, Math. Dept.
October 1969
AD 699-502

*TR-67 Vanderbilt, Dean H.
Controlled Information Sharing in a Computer Utility,
Ph.D. Thesis, 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. Thesis, Sloan School
June 1970
AD 710-011

*TR-69 Gertz, Jeffrey L.
Hierarchical Associative Memories for Parallel Computation,
Ph.D. Thesis, 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,
B.S. & M.S. Theses, EE Dept.
June 1970
AD 711-060

*TR-71 Fiasconaro, James G.
A Computer-Controlled Graphical Display Processor,
M.S. 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. Thesis, Math. Dept.
August 1970
AD 712-069

TR-74 Edelberg, Murray
Integral Convex Polyhedra and an Approach to Integralization,
Ph.D. Thesis, 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. Thesis, EE Dept.
September 1970
AD 713-988

TR-77 Haggerty, Joseph P.
Complexity Measures for Language Recognition by Canonic Systems,
M.S. Thesis, EE Dept.
October 1970
AD 715-134

*TR-78 Madnick, Stuart E.
Design Strategies for File Systems,
M.S. 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. Thesis, 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. Thesis, ME Dept.
January 1971

AD 717-951

*TR-82 Krakauer, Lawrence J.
Computer Analysis of Visual Properties of Curved Objects,
Ph.D. Thesis, EE Dept.
May 1971

AD 723-647

*TR-83 Lewin, Donald E.
In-Process Manufacturing Quality Control,
Ph.D. Thesis, 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. Thesis, 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. Thesis, EE Dept.
June 1971
AD 725-859

TR-87 Thomas, Robert H.
A Model for Process Representation
and Synthesis,
Ph.D. Thesis, EE Dept.
June 1971
AD 726-049

*TR-88 Welch, Terry A.
Bounds on Information Retrieval
Efficiency in Static File Structures,
Ph.D. Thesis, EE Dept.
June 1971
AD 725-429

TR-89 Owens, Richard C., Jr.
Primary Access Control in Large-Scale Time-Shared Decision Systems,
M.S. Thesis, Sloan School
July 1971
AD 728-036

TR-90 Lester, Bruce P.
Cost Analysis of Debugging Systems,
B.S. & M.S. Theses, EE Dept.
September 1971
AD 730-521

*TR-91 Smoliar, Stephen W.
A Parallel Processing Model of
Musical Structures,
Ph.D. Thesis, Math. Dept.
September 1971
AD 731-690

TR-92 Wang, Paul S.
Evaluation of Definite Integrals
by Symbolic Manipulation
Ph.D. Thesis, Math. Dept.
October 1971
AD 732-005

TR-93 Greif, Irene Gloria
Induction in Proofs about Programs,
M.S. Thesis, EE Dept.
February 1972
AD 737-701

TR-94 Hack, Michel Henri Theodore
Analysis of Production Schemata
by Petri Nets,
M.S. Thesis, EE Dept.
February 1972
AD 740-320

*TR-95 Fateman, Richard J.
Essays in Algebraic Simplification
(A revision of a Harvard Ph.D. Thesis)
April 1972
AD 740-132

TR-96 Manning, Frank
Autonomous, Synchronous Counters Constructed Only of
J-K Flip-Flops,
M.S. Thesis, EE Dept.
May 1972
AD 744-030

TR-97 Vilfan, Bostjan
The Complexity of Finite Functions
Ph.D. Thesis, EE Dept.
March 1972
AD 739-678

TR-98 Stockmeyer, Larry Joseph
Bounds on Polynomial Evaluation Algorithms
M.S. Thesis, EE Dept.
April 1972
AD 740-328

TR-99 Lynch, Nancy Ann
Relativization of the Theory of Computational Complexity
Ph.D. Thesis, Math. Dept.
June 1972
AD 744-032

TR-100 Mandl, Robert
Further Results on Hierarchies of Canonic Systems
M.S. 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
M.S. Thesis, EE Dept.
September 1972
AD 749-367

*TR-103 Sekino, Akira
Performance Evaluation of Multiprogrammed Time-Shared Computer Systems
Ph.D Thesis, EE Dept.
September 1972
AD 749-949

TR-104 Schroeder, Michael D.
Cooperation of Mutually Suspicious Subsystems in a Computer Utility
Ph.D. Thesis, 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
M.S. Thesis, EE Dept.
January 1973
AD 756-248

TR-107 Madnick, Stuart E.
Storage Hierarchy Systems
Ph.D. Thesis, EE Dept.
April 1973
AD 760-001

TR-108 Wand, Mitchell
Mathematical Foundations of Formal Language Theory
Ph.D. Thesis, Math. Dept.
December 1973

TR-109 Johnson, David S.
Near-Optimal Bin Packing Algorithms
Ph.D. Thesis, Math. Dept.
June 1973
PB 222-090

TR-110 Moll, Robert
Complexity Classes of Recursive Functions
Ph.D. Thesis, Math. Dept.
June 1973
AD 767-730

TR-111 Linderman, John P.
Productivity in Parallel Computation Schemata
Ph.D. Thesis, EE Dept.
December 1973
PB 226-159/AS

TR-112 Hawryszkiewycz, Igor T.
Semantics of Data Base Systems
Ph.D. Thesis, EE Dept.
December 1973
PB 226-061/AS

TR-113 Herrmann, Paul P.
On Reducibility Among Combinatorial Problems
M.S. Thesis, Math. Dept.
December 1973
PB 226-157/AS

TR-114 Metcalfe, Robert M.
Packet Communication
Ph.D. Thesis, Applied Math., Harvard University
December 1973
AD 771-430

TR-115 Rotenberg, Leo
Making Computers Keep Secrets
Ph.D Thesis, EE Dept.
February 1974
PB 229-352/AS

TR-116 Stern, Jerry A.
Backup and Recovery of On-Line Information in a Computer Utility
M.S. & E.E. Theses, EE Dept.
January 1974
AD 774-141

TR-117 Clark, David D.
An Input/Output Architecture for Virtual Memory Computer Systems
Ph.D. Thesis, 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. Thesis, EE Dept.
February 1974
AD 775-545

*TR-120 Ramchandani, Chander
Analysis of Asynchronous Concurrent Systems by Timed Petri Nets
Ph.D. Thesis, EE Dept.
February 1974
AD 775-618

TR-121 Yao, Foong F.
On Lower Bounds for Selection Problems
Ph.D. Thesis, Math. Dept.
March 1974
PB 230-950/AS

TR-122 Scherf, John A.
Computer and Data Security: A Comprehensive Annotated Bibliography
M.S. 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
B.S. & M.S. Theses, EE Dept.
March 1974
PB 231-365/AS

TR-125 Mark, William S.
A Model-Debugging System
B.S. & M.S. Theses, EE Dept.
April 1974
AD 778-688

*TR-126 Altman, Vernon E.
A Language Implementation System
B.S. & M.S. Theses, Sloan School
May 1974
AD 780-672

TR-127 Greenberg, Bernard S.
An Experimental Analysis of Program Reference Patterns in the Multics Virtual Memory
M.S. Thesis, EE Dept.
May 1974
AD 780-407

TR-128 Frankston, Robert M.
The Computer Utility as a Marketplace for Computer Services
M.S. & E.E. Theses, EE Dept.
May 1974
AD 780-436

TR-129 Weissberg, Richard W.
Using Interactive Graphics in Simulating the Hospital Emergency Room
M.S. Thesis, EE Dept.
May 1974
AD 780-437

TR-130 Ruth, Gregory R.
Analysis of Algorithm Implementations
Ph.D. Thesis, 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
M.S. 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. Thesis, EE Dept.
July 1974
PB 235-283/AS

*TR-134 Ellis, David J.
Semantics of Data Structures and References
M.S. & E.E. Theses, EE Dept.
August 1974
PB 236-594/AS

TR-135 Pfister, Gregory F.
The Computer Control of Changing Pictures
Ph.D. Thesis, EE Dept.
September 1974
AD 787-795

TR-136 Ward, Stephen A.
Functional Domains of Applicative Languages
Ph.D. Thesis, EE Dept.
September 1974
AD 787-796

TR-137 Seiferas, Joel I.
Nondeterministic Time and Space Complexity Classes
Ph.D Thesis, Math. Dept.
September 1974
PB 236-777/AS

TR-138 Yun, David Y. Y.
The Hensel Lemma in Algebraic Manipulation
Ph.D. Thesis, Math. Dept.
November 1974
AD A002-737

TR-139 Ferrante, Jeanne
Some Upper and Lower Bounds on Decision
Procedures in Logic
Ph.D. Thesis, Math. Dept.
November 1974
PB 238-121/AS

TR-140 Redell, David D.
Naming and Protection in Extendible
Operating Systems
Ph.D. Thesis, 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
M.S. & E.E. Theses, EE Dept.
December 1974
AD A003-002

TR-143 Silverman, Howard
A Digitalis Therapy Advisor
M.S. Thesis, EE Dept.
January 1975

TR-144 Rackoff, Charles
The Computational Complexity of Some
Logical Theories
Ph.D. Thesis, EE Dept.
February 1975

*TR-145 Henderson, D. Austin
The Binding Model: A Semantic Base
for Modular Programming Systems
Ph.D. Thesis, 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. Thesis, EE Dept.
February 1975

TR-147 Van De Vanter, Michael L.
A Formalization and Correctness Proof
of the CGOL Language System
M.S. Thesis, EE Dept.
March 1975

TR-148 Johnson, Jerry
Program Restructuring for Virtual Memory Systems
Ph.D. Thesis, EE Dept.
March 1975
AD A009-218

*TR-149 Snyder, Alan
A Portable Compiler for the Language C
B.S. & M.S. Theses, EE Dept.
May 1975
AD A010-218

*TR-150 Rumbaugh, James E.
A Parallel Asynchronous Computer Architecture
for Data Flow Programs
Ph.D. Thesis, EE Dept.
May 1975
AD A010-918

TR-151 Manning, Frank B.
Automatic Test, Configuration, and Repair
of Cellular Arrays
Ph.D. Thesis, EE Dept.
June 1975
AD A012-822

TR-152 Qualitz, Joseph E.
Equivalence Problems for Monadic Schemas
Ph.D. Thesis, EE Dept.
June 1975
AD A012-823

TR-153 Miller, Peter B.
Strategy Selection in Medical Diagnosis
M.S. Thesis, EE & CS Dept.
September 1975

TR-154 Greif, Irene
Semantics of Communicating Parallel Processes
Ph.D. Thesis, EE & CS Dept.
September 1975
AD A016-302

TR-155 Kahn, Kenneth M.
Mechanization of Temporal Knowledge
M.S. Thesis, EE & CS Dept.
September 1975

TR-156 Bratt, Richard G.
Minimizing the Naming Facilities Requiring Protection in a Computer Utility
M.S. Thesis, EE & CS Dept.
September 1975

*TR-157 Meldman, Jeffrey A.
A Preliminary Study in Computer-Aided Legal Analysis
Ph.D. Thesis, EE & CS Dept.
November 1975
AD A018-997

TR-158 Grossman, Richard W.
Some Data-base Applications of Constraint Expressions
M.S. 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
M.S. Thesis, EE & CS Dept.
May 1976
AD A026-688

TR-161 Hack, Michel
Decidability Questions
Ph.D. Thesis, EE & CS Dept.
June 1976

TR-162 Kent, Stephen T.
Encryption-Based Protection Protocols for
Interactive User-Computer Communication
M.S. 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
M.S. & E.E. Theses, EE & CS Dept.
June 1976

TR-164 Reed, David P.
Processor Multiplexing in a Layered Operating System
M.S. Thesis, EE & CS Dept.
July 1976

TR-165 McLeod, Dennis J.
High Level Expression of Semantic Integrity
Specifications in a Relational Data Base System
M.S. 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
M.S. Thesis, EE & CS Dept.
September 1976
AD A034-185

TR-167 Janson, Philippe A.
Using Type Extension to Organize Virtual Memory
Mechanisms
Ph.D. Thesis, 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. Thesis, EE & CS Dept.
December 1976

TR-171 Huber, Andrew R.
A Multi-Process Design of a Paging System
M.S. & E.E. Theses, EE & CS Dept.
December 1976

TR-172 Mark, William S.
The Reformulation Model of Expertise
Ph.D. Thesis, EE & CS Dept.
December 1976
AD A035-397

TR-173 Goodman, Nathan
Coordination of Parallel Processes in the Actor Model of Computation
M.S. Thesis, EE & CS Dept.
December 1976

TR-174 Hunt, Douglas H.
A Case Study of Intermodule Dependencies in a Virtual Memory Subsystem
M.S. & E.E. Theses, EE & CS Dept.
December 1976

TR-175 Goldberg, Harold J.
A Robust Environment for Program Development
M.S. Thesis, EE & CS Dept.
February 1977

TR-176 Swartout, William R.
A Digitalis Therapy Advisor with Explanations
M.S. Thesis, EE & CS Dept.
February 1977

TR-177 Mason, Andrew H.
A Layered Virtual Memory Manager
M.S. & E.E. Theses, EE & CS Dept.
May 1977

*TR-178 Bishop, Peter B.
Computer Systems with a Very Large Address Space and Garbage Collection
Ph.D. Thesis, EE & CS Dept.
May 1977
AD A040-601

TR-179 Karger, Paul A.
Non-Discretionary Access Control for Decentralized Computing Systems
M.S. Thesis, EE & CS Dept.
May 1977
AD A040-804

TR-180 Luniewski, Allen W.
A Simple and Flexible System Initialization Mechanism
M.S. & E.E. Theses, EE & CS Dept.
May 1977

TR-181 Mayr, Ernst W.
The Complexity of the Finite Containment Problem for Petri Nets
M.S. Thesis, EE & CS Dept.
June 1977

TR-182 Brown, Gretchen P.
A Framework for Procecssing Dialogue
June 1977
AD A042-370

TR-183 Jaffe, Jeffrey M.
Semilinear Sets and Applications
M.S. Thesis, EE & CS Dept.
July 1977

*TR-184 Levine, Paul H.
Facilitating Interprocess Communication in a Heterogeneous Network Environment
B.S. & M.S. Theses, EE & CS Dept.
July 1977
AD A043-901

TR-185 Goldman, Barry
Deadlock Detection in Computer Networks
B.S. & M.S. Theses, EE & CS Dept.
September 1977
AD A047-025

TR-186 Ackerman, William B.
A Structure Memory for Data Flow Computers
M.S. Thesis, EE & CS Dept.
September 1977
AD A047-026

TR-187 Long, William J.
A Program Writer
Ph.D. Thesis, EE & CS Dept.
November 1977
AD A047-595

TR-188 Bryant, Randal E.
Simulation of Packet Communication
Architecture Computer Systems
M.S. Thesis, EE & CS Dept.
November 1977
AD A048-290

TR-189 Ellis, David J.
Formal Specifications for Packet
Communication Systems
Ph.D. Thesis, EE & CS Dept.
November 1977
AD A048-980

TR-190 Moss, J. Eliot B.
Abstract Data Types in Stack Based Languages
M.S. 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. Thesis, EE & CS Dept.
January 1978
AD A051-149

TR-192 Niamir, Bahram
Attribute Partitioning in a Self-
Adaptive Relational Database System
M.S. Thesis, EE & CS Dept.
January 1978
AD A053-292

TR-193 Schaffert, J. Craig
A Formal Definition of CLU
M.S. 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
M.S. 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

*TR-197 Baker, Henry Jr.
Actor Systems for Real-Time Computation
Ph.D. Thesis, EE & CS Dept.
March 1978
AD A053-328

TR-198 Halstead, Robert H., Jr.
Multiple-Processor Implementation of Message-Passing Systems
M.S. Thesis, EE & CS Dept.
April 1978
AD A054-009

*TR-199 Terman, Christopher J.
The Specification of Code Generation Algorithms
M.S. Thesis, EE & CS Dept.
April 1978
AD A054-301

TR-200 Harel, David
Logics of Programs: Axiomatics and Descriptive Power
Ph.D. Thesis, EE & CS Dept.
May 1978

TR-201 Scheifler, Robert W.
A Denotational Semantics of CLU
M.S. Thesis, EE & CS Dept.
June 1978

TR-202 Principato, Robert N., Jr.
A Formalization of the State Machine
Specification Technique
M.S. & E.E. Theses, EE & CS Dept.
July 1978

TR-203 Laventhal, Mark S.
Synthesis of Synchronization Code for
Data Abstractions
Ph.D. Thesis, EE & CS Dept.
July 1978
AD A058-232

TR-204 Teixeira, Thomas J.
Real-Time Control Structures for Block
Diagram Schemata
M.S. Thesis, EE & CS Dept.
August 1978
AD A061-122

TR-205 Reed, David P.
Naming and Synchronization in a Decentralized
Computer System
Ph.D. Thesis, EE & CS Dept.
October 1978
AD A061-407

TR-206 Borkin, Sheldon A.
Equivalence Properties of Semantic Data
Models for Database Systems
Ph.D. Thesis, EE & CS Dept.
January 1979
AD A066-386

TR-207 Montgomery, Warren A.
Robust Concurrency Control for a Distributed
Information System
Ph.D. Thesis, EE & CS Dept.
January 1979
AD A066-996

TR-208 Krizan, Brock C.
A Minicomputer Network Simulation System
B.S. & M.S. Theses, EE & CS Dept.
February 1979

TR-209 Snyder, Alan
A Machine Architecture to Support an Object-Oriented Language
Ph.D. Thesis, 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
M.S. Thesis, EE & CS Dept.
April 1979

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. Thesis, 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. Thesis, 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

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

Copies of all reports with AD and PB numbers listed in Publications may be secured from the National Technical Information Service, Operations Division, Springfield, Virginia, 22151. Prices vary. The AD or PB number must be supplied with the request.

* Out of Print reports may be obtained from NTIS if the AD number is supplied (see above). Out of Print reports without an AD or PB number are unobtainable.

OFFICIAL DISTRIBUTION LIST

Director — 2 copies
Defense Advanced Research
Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

Attention: Program Management

Office of Naval Research — 3 copies
800 North Quincy Street
Arlington, Virginia 22217

Attention: Mr. Marvin Denicoff
Code 437

Office of Naval Research — 1 copy
Resident Representative
Massachusetts Institute of Technology
Building E19-628

Attention: A. Forrester

Director — 6 copies
Naval Research Laboratory
Washington, DC 20375

Attention: Code 2627

Defense Documentation Center — 12 copies
Building 5 Cameron Station
Alexandria, Virginia 22213

Office of Naval Research — 1 copy
Branch Office/Boston
Building 114, Section D
666 Summer Street
Boston, Massachusetts 02210

DATE
FILMED

-8