ANDREW: A DISTRIBUTED PERSONAL COMPUTING ENVIRONMENT

The Information Technology Center (ITC), a collaborative effort between IBM and Carnegie-Mellon University, is in the process of creating Andrew, *a prototype computing and communication system for universities. This article traces the origins of Andrew, discusses its goals and strategies, and gives an overview of the current status of its implementation and usage.*

JAMES H. MORRIS, MAHADEV SATYANARAYANAN, MICHAEL H. CONNER, JOHN H. HOWARD, DAVID S. H. ROSENTHAL, and F. DONELSON SMITH

In October 1982, Carnegie-Mellon University (C-MU) and IBM agreed to create the Information Technology Center (ITC), an organization consisting of about 30 people, with the task of designing and developing computing technology to support C-MU's needs by the fall of 1986. Ten members of the organization would be IBM employees on assignment, and the system—named *Andrew*, after two benefactors of C-MU, Andrew Carnegie and Andrew Mellon—would be based partly on IBM hardware.

Several other organizations were assigned formidable and complementary activities: The Computation Center would deploy and maintain the system, the Center for Development of Educational Computing would facilitate the production of course-related software, and IBM's Academic Information Systems independent business unit would develop and market related and derived IBM products.

The project will affect university education in four main areas:

- *Computer-aided instruction.* For many years, some pioneering institutions have used computers to deliver instruction. Perhaps the largest effort is the Plato Project [26] at the University of Illinois where over 10,000 hours of course material have been developed. We expect the presence of ubiquitous and powerful graphics workstations to greatly stimulate efforts to develop computer-aided instruction—producing computer-animated demonstrations is a new art, and the university is well poised to develop it.
- *Creation and use of new tools.* A research university like C-MU is just as devoted to improving the methods of professional work as to teaching them. A professor's research on a new tool can often be aided by teaching it, while students form a good set of users and sometimes assist in the creation of new software. Virtually every department at C-MU is pursuing the development of computer tools. Beyond the expected activity in engineering and science, there are also developments in writing, historical research, social science, music, painting, psychology, and many other fields as well.
- *Communication.* With every member of a university community plugged into a smoothly operating communication system, one can expect far-reaching effects [10, 27, 31]. Class discussions held on a computer bulletin board will last longer, involve more participants, and allow time for more reflection and analysis. The use of graphics, formula manipulation, and automatic typesetting can improve the form and, indirectly, the content of much of our communication. We do not expect computer-mediated communication to supplant the more traditional methods, but it will broaden and deepen the community's ability to communicate.
- *Information access.* A mark of tomorrow's professional will be the ability to navigate in large information repositories. A university's primary database is its library of books and journals, and our communication system will provide better access to it. In addition, it will provide access to the growing number of worldwide databases. Finally, we expect universities to develop extensive new databases devoted to particular courses and areas of specialization.

(See a recent paper from M.I.T.'s Project Athena [3] for a more extensive catalog of possibilities.)

THE TECHNICAL FORERUNNERS OF ANDREW

The Xerox Alto System [11] has been a powerful inspiration for our system, which is based on four key components: *personal computers, raster graphics, high bandwidth networks,* and *time-sharing file systems.*

Personal Computing

The era of personal computing was ushered in by the introduction of stand-alone microprocessor-based computers by companies such as Apple and IBM. The identifying characteristic of this class of machines was their cost—they had to be affordable to small businesses and individuals. This constraint and the available technology dictated the architecture, hardware implementation, and software aspects of these machines; and thus floppy disks, 8-bit data paths, character displays, and Basic were the order of the day. From a user's point of view, the major real difference between using such a machine and using a time-sharing system was the fact that the performance of the former was constant and predictable, unaffected by the activities of other users.

Raster Graphics

The designers of the Alto [33] realized that, once a terminal was replaced with a powerful computer, higher bandwidth between the computer and its user could be exploited to present information in a significantly more appealing way. Integral to the design was a pixel-addressable display mapped into main memory, and a pointing device (a mouse). The software for this machine treated the screen as a two-dimensional collage of text and graphical images rather than a one-dimensional string of text. Small graphical icons were used to symbolize actions or states, and mouse movements and button clicks rather than key strokes were used to communicate with the machine. The model of human–machine communication first demonstrated in this project has now come to be accepted as highly desirable, particularly for novices.

High Bandwidth Communications

In moving away from time-sharing to personal computing, it would indeed be unfortunate if users lost the ability to communicate among themselves. Normal asynchronous communication at kilobit rates has been a bottleneck when transferring files or doing anything besides interacting with a time-sharing system. Local-area networks (LANs) provide megabit rate communication. Originating with the Ethernet [14] in the Alto project, the linking of workstations by a LAN has become standard practice in laboratory environments. LANs make possible the shared use of relatively expensive peripherals such as large disks, laser printers, and tape drives, and small files can be transferred between machines with hardly noticeable delays.

Time-Sharing File Systems

An unexpected benefit of time-sharing was the use of computers as a vehicle for communication among users. Sharing of information via a common file system is now taken for granted. In fact, there are many users for whom the communication and information-sharing aspects of a computer are far more important than its computational capability.

In a time-sharing environment, cooperation between users is particularly simple because of the existence of common logical name spaces. For example, two users who are sharing a file refer to it using the same name; the physical locations at which they are logged in are immaterial. As another example, mail from one user to another need only specify the recipient's name; no information need be given about the terminal at which the latter will log in to read mail.

Although a networked personal computing environment provides connectivity, it does not automatically imply the same ease of sharing. Each workstation in such an environment has a unique network address that has to be specified when accessing files. In addition, explicit user actions are usually required to achieve sharing: Before using a file, a user must run a program to transfer it to the network node where he or she is currently located; changes made by that user are not visible to other workstations until the modified file has been transferred to them or to a central repository. User mobility is also limited—one cannot create a file at one workstation, walk to another workstation, and access the file with the effortlessness that is possible when using geographically separated terminals of a time-sharing system.

BASIC DECISIONS

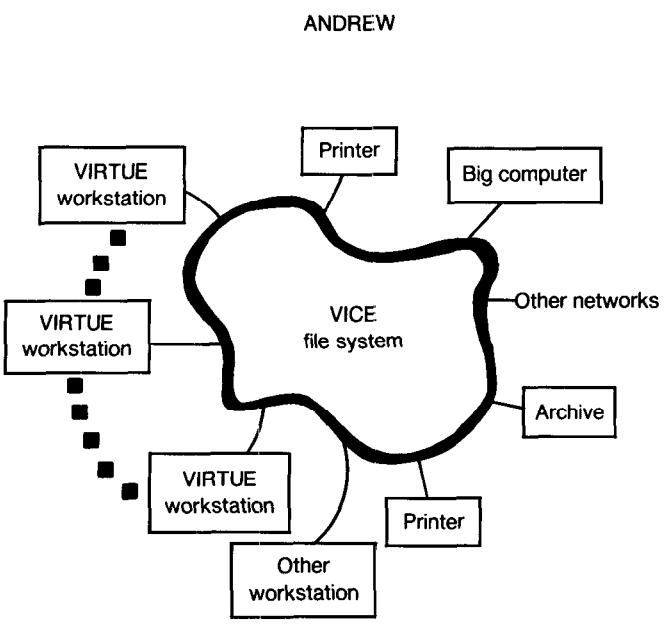
The computing paradigm envisioned in Andrew is a marriage between personal computing and time-sharing. It incorporates the flexibility and visually rich user–machine interface made possible by the former, with the ease of communication and information-sharing characteristic of the latter. This model is depicted in Figure 1 (p. 186).

The VICE–VIRTUE interface has two distinctive characteristics:

1. *It is a relatively static programming interface.* Enhancements to this interface will typically be made in an upwardly compatible manner. This allows advances in technology to be taken advantage of, without systemwide trauma. A new type of workstation will require some software development to integrate it with VICE. However, existing workstations will not be affected in any way. In the long run, therefore, one can expect a situation where nonhomogeneous workstations are attached to VICE, but will share its resources in a uniform manner.
2. *It is the boundary of trustworthiness.* All computing and communication elements within VICE may be assumed to be secure. This guarantee is achieved through physical and administrative control of VICE computers. No user programs are executed in VICE, and it is therefore an internally secure environment. Workstations, however, are owned by individuals who are free to modify their hardware and software in any way they wish. Encryption-based authentication and transmission will be used to ensure the security of VICE–VIRTUE communication.

Workstation Hardware

The model we chose for our personal computer was much more powerful than any existing one. The C-MU



The amoebalike structure in the middle, called VICE, is a collection of communication and computational resources serving as the backbone of a user community. Individual workstations, called VIRTUEs, are attached to VICE and provide users with the computational cycles needed for actual work as well as a sophisticated user–machine interface. (VICE stands for "Vast, Integrated Communications Environment"; VIRTUE, for "Virtue is reached through UNIX® and EMACS.")

FIGURE 1. VICE and VIRTUE

Computer Science Department had created a carefully thought-out machine specification for its Spice Project [4] that included such advanced requirements as a million-instruction-per-second processor, a bit-map display with a million pixels, virtual memory, a megabyte of real memory, and a LAN connection. They predicted, accurately, that such machines could be had for $10,000 in the mid eighties; so it seemed that subsequent price reductions would make it an affordable machine for students a few years later.

Although we agreed on the kind of workstation that was needed, IBM had no such product. Laboratory prototypes existed, but were not ready to be used at C-MU. We chose the SUN workstation as our development machine since it had most of the characteristics of a Spice machine. At the time (April 1983), SUNs had barely started production, so we suffered through some of their growing pains, but have been generally happy with the decision.

A decision we have yet to make relates to the use of disk storage. There are many reasons to dislike workstations with hard disks: They are expensive, power consuming, noisy, and subject to failure. A floppy disk-based system was obviously unable to support virtual memory, but the SUN system and others had proved that it was feasible to use a LAN to deliver pages to a workstation from a dedicated, shared disk server. Nevertheless, we have not committed to diskless workstations as a basic strategy because of several considerations:

- It would have led to a fundamentally less-robust system design; if the network were down or overloaded, the workstations would be unusable. Diskless machines greatly increase traffic on the network, a central resource that might be hard to augment.
- When the cost of a disk server was amortized over the machines it could support well—about 10 for SUNs—the costs were about the same as providing each workstation with its own disk.
- It seemed unrealistic for an individual to purchase a machine that would only work when it was on the C-MU campus. Some of the C-MU community works off campus, out of reach of high bandwidth communications; and students who live on campus are actually in residence for barely half the year.
- Paging over an open network theoretically precludes most schemes for guaranteeing privacy.
- The ITC runs a mixture of workstations, some with disks and some without. Workstations with disks are much preferred because the performance is perceived as better and less variable.

Operating System

We chose the Berkeley UNIX operating system [20] for several reasons: It was a well-defined standard, had several advanced features, and was well liked by many developers in the university environment. One of the most important features was that it was portable: This made it possible to develop Andrew on one machine and move it to others. C-MU's Accent operating system [18] had some superior technical features, but was not fully developed and not demonstrably portable. The MS-DOS environment, which was to become the de facto standard for commercial personal computing, did not offer a programmers' environment that was as good; it lacked multiprocessing features and virtual memory, and was portable only among PC clones. In retrospect, this decision was a good one in that it has allowed rapid development and gives our system access to a variety of workstations. Nevertheless, it has made the system rather large and expensive relative to the goal of providing it on inexpensive student workstations. Our hope is that the rising tide of hardware technology will solve this problem if we can avoid raising our ambitions along with it.

In choosing such a powerful workstation and a virtual-memory, multiprocessing operating system, we distanced ourselves from the mainstream personal computing world—IBM PCs and Apple MacIntoshes. Given IBM's sponsorship of the project, it might seem obvious that we should have based our system on IBM PCs. Why didn't we? First, the PC does not have very good support for raster graphics, one of our key criteria. Second, while the commercial world of PCs is character-

UNIX is a trademark of AT&T Bell Laboratories.

ized by a small number of vendors producing software for a huge, nonprogramming consumer base, we expect a high percentage of C-MU's population to engage in software development at some level. Therefore, programmer comforts such as virtual memory and a good programming environment are important. Many of the faculty at C-MU use powerful workstations in their research; it seems more likely that good educational software and tools will result if the students use a similar machine. Finally, it seems that the personal computing industry will move to computers and operating systems with just the technical characteristics we have chosen. We expect to see a convergence of the personal computer and the engineering workstation in the next few years. When this happens, we expect all the commercially available software to be available to our system as well.

Communications Paradigm

The ITC aspires to interconnect over 5000 workstations. An academic environment requires a large amount of information sharing; a high degree of user mobility between dormitories, faculty offices, libraries, and laboratories is essential. It was therefore important to choose the central communications model carefully.

We chose to emulate a time-sharing file system in which each workstation would appear to be sharing a single large file system. This model was easy to grasp and had clear advantages over the ambitious ones presented by the telephone system and electronic mail systems. We considered technically more ambitious paradigms such as a distributed database system, but rejected them since the pace and content of communication at a university do not require the interconnected structure of most database systems. Most of the information used by a university is in the form of papers, memos, and computer programs. Thus, it seemed acceptable to make the unit of shareable information a file and to record changes centrally every few seconds.

VICE provides a common name space for files. Users may thus access files in a uniform manner regardless of the specific workstations at which they are logged in, or where the files were created originally. Most other shared facilities, such as mail, bulletin boards, and printing, can be built on top of VICE, obviating the need for machine or location-specific information. One requests a service by leaving a request file in a designated directory. Software in each VIRTUE workstation makes these facilities of VICE appear as a transparent extension of that workstation's operating system.

C-MU's relatively small campus and localized student residences allowed us to choose high-bandwidth communications (4 to 10 Mbits per second) as the normal mode of operation. Nevertheless, we find it necessary to support slower communications as a secondary goal since faculty and graduate students often work off campus.

System Components

Figure 2 shows the major components of Andrew. The ITC development effort has focused on three of these components: network communication, the shared file system, and the user interface on an individual workstation. The next three sections of this article examine each of these topics.

NETWORK COMMUNICATION

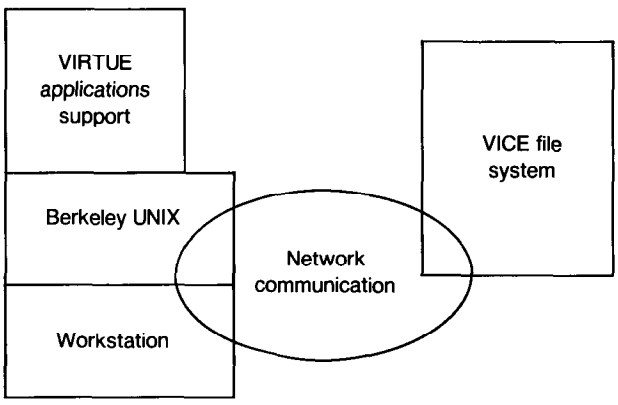
The design strategy that has been chosen exploits locality of reference to reduce network and server utilization. Viewed at a finer granularity than Figure 1, VICE is composed of a collection of semiautonomous clusters connected together by a backbone LAN. Figure 3 (p. 188) illustrates such an interconnection scheme.

The wiring of the campus with the IBM cabling system is expected to be completed by the end of 1986. The plans call for the use of a token ring network developed at IBM [28] and conforming to the IEEE Standard 802 [30].

The Current Reality

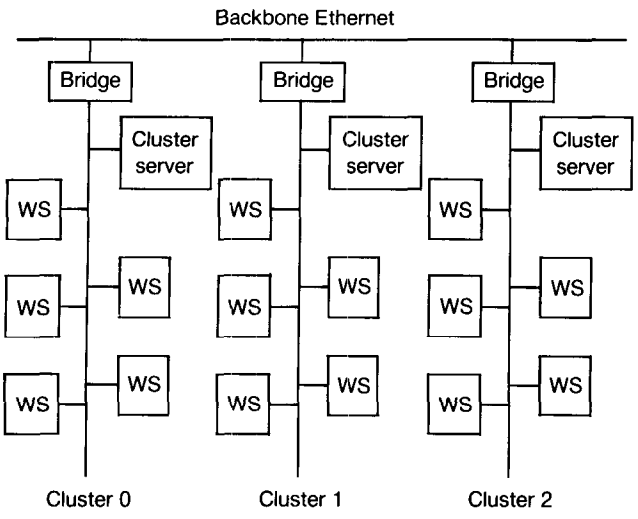
C-MU already had a large number of Ethernets and a few ProNets installed by 1983. As it became apparent that the plan outlined above was going to take considerable time to implement, we decided to build our system on top of these existing networks, interconnecting them and buttressing them where needed. Aside from reducing the apparent effort, it would give an Andrew workstation access to many other machines on the campus immediately without having to construct gateways.

The facility for interconnecting networks was already at hand in the form of *router* machines [2]—PDP/11s programmed by the computer science department—that allowed physical interconnection of different network technologies and supported a simple broadcast



The system structure of Andrew was chosen to allow multiple options to be tried for each component, either simultaneously or through time. Any workstation that can support Berkeley UNIX can be used to run VIRTUE. There are no hard dependencies between the file system and the user interface; each has been used without the other. Multiple network implementations are possible.

FIGURE 2. The Components of Andrew



Each cluster consists of a collection of workstations and a representative of VICE called a cluster server. The *bridges* that connect individual clusters to the backbone serve both as routers and as traffic filters. The routing capability of these elements provides a uniform network address space for all nodes, obviating the need for any end-to-end routing by servers and workstations.

FIGURE 3. VICE Fine Structure

protocol to allow one machine to rendezvous with any other.

The Computation Center had already managed the installation of fiber-optic cables connecting all the academic buildings. Using various products from DEC, Ungermann-Bass, and American Photonics, we extended several of the Ethernets over the fiber so that they could have a representative router machine in a common building, the University Computation Center. By connecting those routers to a common Ethernet, we quickly created an internetwork containing over 600 machines. Figure 4 shows the current campus internet.

Remarkably, this network came about through the voluntary cooperation of several different departments and centers of the university. In a world where it is not unheard of for single academic departments to disagree about a basic network, C-MU has been particularly fortunate in being able to manage this. Further information about the network may be found in [12] and [13].

This ad hoc networking tactic has allowed us to get started quickly; Andrew workstations can communicate with many other computers at C-MU today. Nevertheless, it will not suffice for the long term. Network reliability is often perceived to be a problem, and troubleshooting is complicated because many different organizations own pieces of the net and routing machines. The simple broadcast method of machine rendezvous will need refinement lest it exhaust some resource. Even a single Ethernet can present mysteries if it has not been laid out with a plan for maintenance. On the brighter side: This tactic exposed us to the maintenance implications of a campuswide network early in the project.

Network Protocols

Experience with networking over the last decade has shown the importance of using standardized communication protocols for intermachine communication. Precisely what protocols are used is far less important than the fact that all interconnected machines use the *same* protocol.

Prior to the inception of Andrew, many of the departmental mainframes at C-MU used the DARPA Internet Standard protocols [7] for communication. Owing to its sponsorship by the U.S. Department of Defense, this protocol family has become the lingua franca of the Arpanet user community, of which C-MU is an active member. Further, the Berkeley UNIX operating system already has implementations of these protocols built into it. Consequently, we adopted the TCP/IP protocol family as our standard.

High-level communication between VICE and VIRTUE is based on a client-server model using remote procedure calls (RPCs) for transfer of data and control [15]. An RPC subroutine package has been implemented on top of the Internet protocols [22]. The distinctive features of this package are the following:

1. The transfer of bulk data objects, such as files, as side effects of RPCs. This capability is used extensively in the file system for caching of files at workstations.
2. Built-in authentication facilities that allow two mutually suspicious parties to exchange credentials via a three-phase encrypted handshake. This mechanism will be used by servers in VICE to authenticate users.
3. Optional use of encryption for secure communication, using session keys generated during the authentication handshake.

In order to communicate with various future IBM systems, we implemented SNA (in particular, Logical Unit Type 6.2 [9]) under UNIX. We currently use this package to communicate with IBM 3820 laser printers.

THE SHARED FILE SYSTEM

Network file systems have been the subject of investigation in a number of projects over the last few years [1, 17, 29]. These designs have typically been intended for networks with at most a few hundred nodes. With our ambition to span an order of magnitude more nodes, we felt it essential to approach the design from first principles. We had little confidence that an adaptation of an existing design would prove adequate to the task.

The description of the Andrew file system is in two parts. Pages 189–191 describe the basic architecture, whose rationale has been discussed elsewhere [23]. Pages 191–194 then provide details of the implementation and usage experience with this system. More details on this aspect of Andrew may be found in [34].

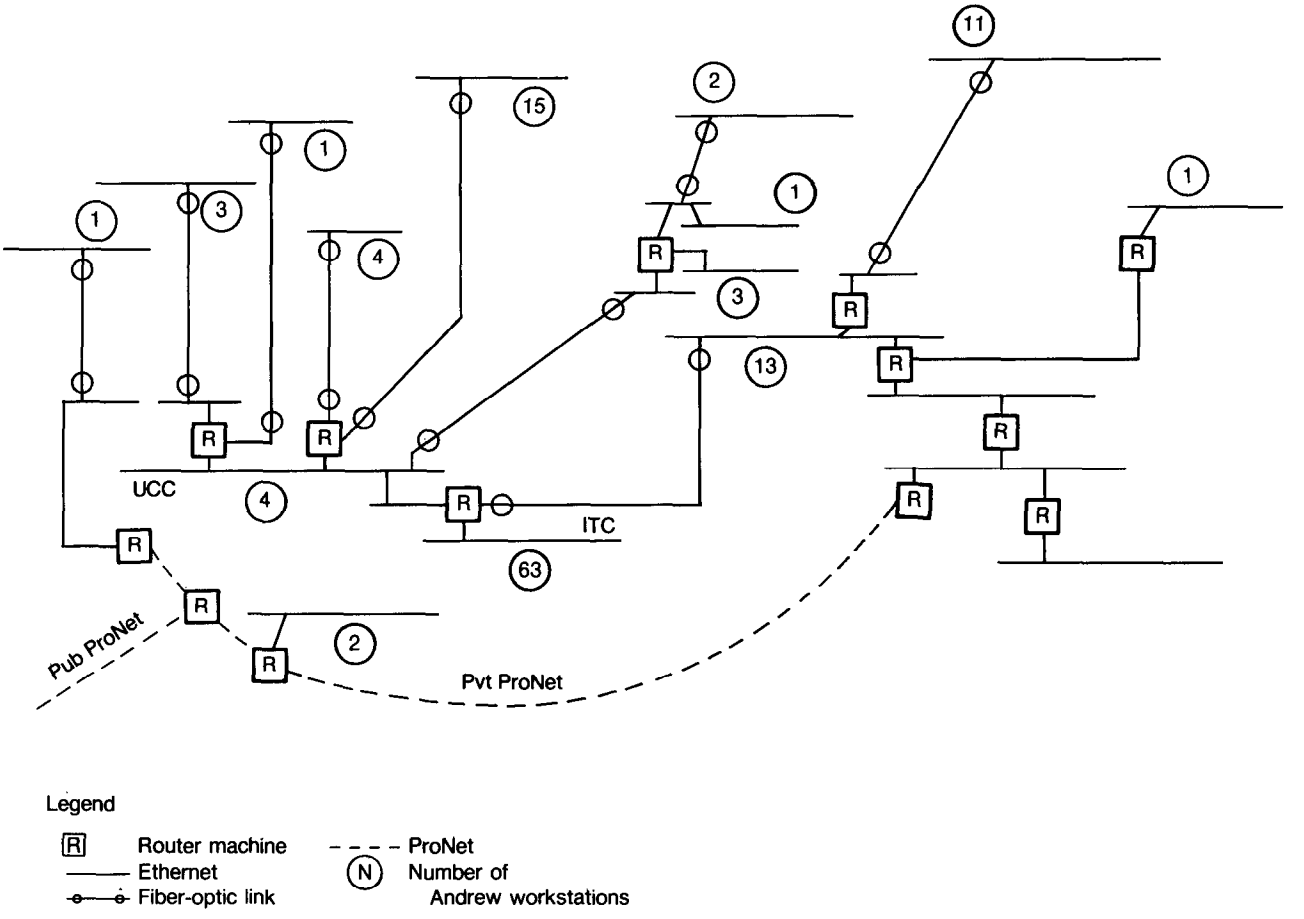
Naming

From the point of view of application programs on workstations, the space of file names is partitioned into two subspaces: *local* and *shared.* Figure 5 (p. 190) illustrates this partitioning. In practice, almost all files accessed by users are in the shared name space. Consequently, users can move at will from one workstation to another and continue to see a consistent image of their files. Both the local and shared name spaces are hierarchically structured and are similar to a time-sharing UNIX file system. In UNIX terminology, the local name space is the *root file system* of a workstation, and the shared name space is *mounted* on the node "/cmu" during workstation initialization. Figure 6 (p. 190) depicts this situation.

UNIX expects to find system files in its local name space. Through the use of symbolic links, however, we have been able to place the vast majority of system files in the shared name space. For example, on a SUN workstation, the local directory */usr/local/bin* is a symbolic link to the remote directory */cmu/unix/sun/usr/local/bin*; on a VAX, */usr/local/bin* is a symbolic link to */cmu/unix/vax/usr/local/bin*. In this way, accesses to most common system files are automatically translated to remote accesses. This greatly reduces the amount of disk space required locally and simplifies the distribution of new releases of system software. As indicated in the example, symbolic links are also of value to us in supporting diversity in workstation hardware.

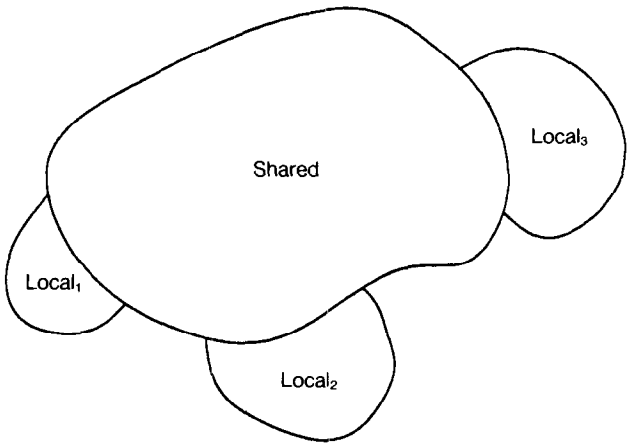
Intercept and Caching

Entire files are cached on demand at workstations. When an application program makes a system call to open a file, the request is first examined by the work-



The current campus internet contains 17 Ethernets and 2 ProNet rings linked by interbuilding fiber-optic cables and routing computers. Over 600 computers can communicate on this internet using DARPA protocols, although only 120 are VIRTUE workstations. The backbone is the net marked UCC. Two cluster servers on the net marked ITC serve about 60 machines attached to that net. Four cluster servers on the backbone support another 60 workstations scattered over all the other nets.

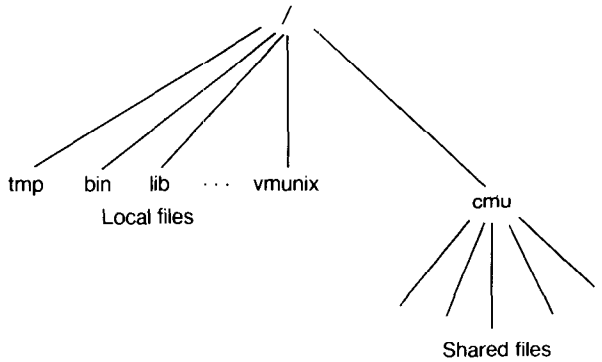
FIGURE 4. The Current C-MU Internet



The shared file name space is identical on all workstations and, though distributed across multiple file servers, appears totally homogeneous to application programs. Since file names do not contain the identity of servers, there is no means by which an application program can determine which file server is storing a particular file. The local name space is small, is distinct for each workstation, and contains files that are either essential for workstation initialization or are temporary files such as those containing intermediate output from compiler phases.

FIGURE 5. Shared and Local Name Spaces

station operating system to determine whether the file is local or shared. In the former case, the open request is satisfied exactly as in a stand-alone system. For a shared file, the request is relayed to a local process,



In UNIX terminology, the local name space is the *root file system* of a workstation, and the shared name space is *mounted* on the node "/cmu" during workstation initialization. Since all shared file names generated on the workstation have "/cmu" as a prefix of their path name, it is trivial to disambiguate between local and shared files.

FIGURE 6. A Workstation's View of the File System

Venus, running on behalf of the file system. Venus manages the local cache and the communication with the remote file server. For an open of a shared file, Venus checks the cache for the presence of a valid copy. If such a copy exists, the open request is treated as an open request to the cached copy. If the file is not present in the cache, or if the copy is not current, a fresh copy is fetched from the appropriate file server. All these actions are transparent to application programs: They merely perform a normal file open.

After a file is opened, individual read and write operations on a shared file are directed to the cached copy: No network traffic is generated on account of such requests. On a close request, the cached copy is first closed as a local file; if it has been modified, the updated copy is then transmitted to the appropriate file server. The cache thus behaves as a write-through cache on closes.

The caching mechanism allows complete mobility of users with minimum performance penalty. If users place all their files in the shared name space (the default), their workstations become "personal" only in the sense that they are owned by them. Users can move to any other workstation attached to VICE and use it exactly as they would use their own workstation. The only observable difference would be an initial performance degradation as the cache on the new workstation is filled with the users' working sets of files.

The caching of entire files, rather than portions of a file, also has a beneficial effect on performance. Network overheads are minimized because servers are contacted only on file opens and closes, and not on individual reads and writes. Cache management on workstations is also simplified, since there are far fewer files than pages of files. To be successful, whole-file transfer requires that each workstation possess adequate secondary storage to cache a typical user's working set of files. Whether this is provided by a disk physically associated with the workstation or is provided by a disk server is an issue that is orthogonal to the design presented here. For reasons mentioned earlier, we prefer to use workstations with local disks.

Inevitably, there are some files that are far too large to fit in workstation caches. These are typically databases, such as the on-line card catalog of the university library. The current design does not address this class of files; separate mechanisms for accessing such databases have to be developed. Except in such cases, actual usage experience has shown that the need to cache entire files is not a problem. We have been able to accommodate files up to a few megabytes in size without any serious difficulties. Studies of file usage patterns in real systems have, in fact, shown that most files tend to be small [16, 21].

Data Location and Replication

Each cluster server in VICE runs a file server process that supports operations such as storing and retrieving files in response to requests from Venii on workstations. The hierarchical file name space is partitioned into disjoint subtrees, and each such subtree is served

by a single server, called its *custodian*. Storage for a file, as well as servicing requests for it, is the responsibility of the corresponding custodian. Changing the custodian of a subtree is a relatively heavyweight operation: The design is predicated on the assumption that such changes do not occur on a minute-to-minute basis.

Certain subtrees that contain frequently read, but rarely modified, files may have read-only replicas at other cluster servers. Such read-only copies are created by a process called *cloning*, initiated by system administrators. Read-only copies are typically created for system files, in order to enhance availability and to evenly distribute server load.

Each server contains a copy of a fully replicated location database that may be queried by Venii to ascertain the custodian of any file. The size of this replicated database is relatively small because custodianship is on a subtree basis: If all files in a subtree have the same custodian, there need only be an entry for the root. The location database changes relatively slowly for two reasons. First, most file creation and deletion activity occurs at depths of the naming tree far below that at which the assignment of custodians is done. Second, reassignment of custodians is infrequent and is initiated via administrative procedures. Consequently, a specialized propagation mechanism that slowly updates custodianship information at all servers is feasible.

For performance reasons, the assignment of custodians to files is done in a way that maximizes the probability that a user's workstation and custodian cluster server are on the same cluster. Faculty members, for instance, would be assigned a custodian on the same cluster server as the workstation in their offices. This assignment does not affect their mobility, since workstation caching would allow them to transparently access their files from any other cluster.

Security

Security is a matter of serious concern to us since individuals may tamper with the hardware and software on workstations they own, and since it is not feasible to guarantee the integrity of an interconnecting network that spans the entire campus. Consequently, our design is not predicated on the trustworthiness of either the workstations or the network. In this respect, Andrew is fundamentally different from other distributed file systems reported in the literature.

We attempt only to safeguard against the unauthorized release or modification of information and do not attempt to prevent instances where legitimate users are denied resources. The latter situation can arise, however, when malicious users modify their workstation to flood the network with packets. We believe that peer pressure and social mores are the only effective deterrents in such situations.

When a user initiates activity at a workstation, VIRTUE authenticates itself to VICE on behalf of that user. After authentication, all future communication on behalf of that user is encrypted with a key generated at the beginning of each session. Authentication and secure transmission are supported by the underlying RPC package. At the present time, these functions are operational, but await integration. We are also awaiting the installation of hardware encryption devices on the workstations. Encryption, which we believe to be an indispensable building block for secure distributed computing, has been made relatively inexpensive by VLSI technology.

VICE uses access lists to protect data stored in it. Entries on an access list are from a protection domain consisting of *users* and *groups*, which are collections of users and other groups. Information about users and groups is stored in a protection database that is replicated at each cluster server. The users' rights on a protected object are the union of the rights specified for all the groups they belong to, either directly or indirectly. Access lists are associated only with directories. Files within a directory may be individually protected against access or modification, but it is not possible to specify selective access to different individuals.

Mail, Printing, and External Communication

The basic architecture of the file system has simplified support for services such as mail and printing. There is no need for special servers to spool anything as long as the sender and receiver of information are both clients of VICE. The mail-transport mechanism is trivial. Each user has a subdirectory called "mailbox" of his or her home directory. Sending the user mail simply consists of storing a file in the mailbox. Although it is clear that one can build an excellent mail system without basing it on a file system [5, 6], we feel that interleaving its functions with a file system will make it more useful. Similarly, spooling a file for printing on a particular printer consists of putting a directive in a subdirectory it owns, and pointing to a file in one's own directory. The flexible, access-list-based protection system makes these operations possible. For example, one's mailbox allows anyone to insert files, but allows only the owner to read files. Official bulletin boards can be achieved by allowing anyone read access, but only officials are allowed write access, to a particular subdirectory.

This simple structure is of no help, however, when one wants to communicate with non-Andrew users at C-MU or elsewhere; we use standard file transfer programs (FTP) and mail gateways to cope. For example, to store a file in VICE from any machine on campus one can perform an FTP to any Andrew workstation, assuming one has a valid password.

Implementation and Experience

The Andrew file system has gone through the normal developmental stages of architecture, prototype, experience, and refinement. In this section, we describe the evolution of the Andrew file system, paying particular attention to our experiences with it, and the lessons we have learned from its design and implementation.

The high-level architecture and key design decisions such as caching and whole-file transfer that were presented earlier were determined quite early in the design. We built a prototype, VICE-I, to determine the viability of this architecture. This system was deployed

to a user community of about 400 users with access to about 100 workstations. Based on our experience with VICE-I, we refined the basic architecture and produced a completely new implementation called VICE-II. Both VICE-I and VICE-II were built on top of the UNIX operating system; but, because users cannot log in to file servers directly, we retain the option to reimplement it in any environment we like. We have used SUNs and VAXes as file servers, each with two or three 400-Mbyte disks.

The following section describes the implementation of VICE-I, while the section following that discusses what we learned from it. The succeeding sections will then describe the implementation and status of VICE-II.

VICE-I Implementation. In VICE-I, a client would rendezvous with a server process listening at a well-known network address on a cluster server. This process then forked a dedicated process to deal with all future requests from the client. The dedicated process vanished when its client terminated the connection to it. In steady state, therefore, a VICE-I cluster server operated with at least as many server processes as there were active clients. Since UNIX does not allow sharing of address spaces between processes, locking was implemented by a dedicated lock server process that serialized requests from the dedicated server processes and maintained a lock table in its address space. All other sharing between the latter processes took place via files in the underlying file system.

Data and VICE status information were both stored in files. Each server contained a UNIX directory hierarchy exactly mirroring the structure of the VICE files stored on it. VICE file status information, such as access lists, was stored in shadow directories called *.admin* directories. The directory hierarchy contained *stub* directories to represent portions of the VICE name space that were located on other servers. The location database that maps files to custodians was thus embedded in the file tree in VICE-I. If a file were not on a server, the search for its name would end in a stub directory that identified the custodian for the file. Below the top levels of the VICE naming tree, files in the same subtree are likely to be located on the same custodian. Hence clients cached path-name prefix information and used this as the basis of a heuristic to direct file requests to appropriate servers.

In VICE-I, the VICE–Venus interface named files by their full path name. There was no notion of a low-level name, such as the *Inode* in UNIX. A rudimentary form of read-only replication, restricted to the top-most levels of the VICE name tree, was present. Each replicated directory had a single server site to which all updates were directed. An asynchronous slow-propagation mechanism reflected changes made at this site to the read-only replicas at all other sites.

All cache entries were considered suspect in VICE-I. Before using the cached copy of a file, Venus would verify that its time stamp matched that of the copy on the custodian. Each file open thus resulted in at least one interaction with a server, even if the file were already in the cache.

Experience with VICE-I. VICE-I was used for nearly a year, eventually expanding to encompass six servers and about 100 workstations. The goals of location transparency and user mobility were met unequivocally and have proved to be addictive. We would now find it difficult to put up with an environment where individuals were tied to specific workstations, or where we had to remember which machine a particular file resided on. Our initial apprehensions about relying solely on caching and whole-file transfer proved to be baseless. Application code compatibility was also met to a very high degree, and almost every UNIX application program was able to use VICE files without recompilation or relinking.

Although our experience with VICE-I was mostly positive, we ran into some problems that we had not anticipated. Probably the single biggest surprise was the frequency of *stat* system calls made by applications. On a standard file system, this is a relatively cheap call. Repeated stats of the same file usually involve no disk activity, since UNIX caches recently read disk blocks in memory. In VICE-I, however, every stat involved an RPC to a server to validate the appropriate cache entry or to fetch it. Under conditions of heavy load, this caused rather annoying performance degradation.

Another annoyance was the inability to rename directories in VICE. This turned out to be a consequence of our using path names in the VICE–Venus interface. The absence of a low-level identifier that was visible to Venus and that remained invariant across a directory rename meant that it would not always be possible to unambiguously answer cache validation requests on certain files after the rename. Another heavily used UNIX feature missing from VICE-I was the ability to use symbolic links. This was a subtle consequence of our decision to resolve path names in VICE rather than Venus.

Measurements indicated that an average cache hit ratio of over 80 percent was attained during normal use. Server CPU utilization tended to be quite high, averaging nearly 40 percent over an 8-hour working day on the most heavily loaded servers. Disk utilization tended to be lower, averaging about 14 percent. Short-term averages were, of course, much higher. A histogram of calls received by servers in actual use showed that cache validation calls accounted for over 65 percent of the total. Calls to fetch file status information contributed about 27 percent, while calls to fetch and store files accounted for 4 percent and 2 percent, respectively. These four calls thus encompassed more than 98 percent of the calls handled by servers.

VICE-I turned out to be a rather difficult system to operate and maintain. Our decision to use a dedicated process per client on each cluster server caused various resource limits to be exceeded on a number of occasions, effectively crashing the cluster server. It also resulted in excessive context switching overhead and in

high virtual-memory paging demands. It did, however, have the virtue of simplicity and resulted in a relatively robust system because the failure of an individual server process affected only one client. The RPC package was built on top of the reliable byte-stream abstraction provided by TCP. Although this yielded a simple implementation, it caused TCP-related resources to be exceeded on occasion, thereby denying service to new clients. Our decision to embed the file location database in stub directories in the VICE naming tree made it difficult to move users' directories between servers. Finally, Venus based its cache replacement algorithm on the total number of files in the cache rather than on the total size of these files. This was done primarily for ease of implementation and worked well most of the time. Sometimes, however, a user's cache would fill his or her local disk to capacity, resulting in a failure mode that nonexpert users found particularly disconcerting. The effects of this problem were particularly severe when it involved an unattended workstation such as a print or mail spooler.

Based on our experience with VICE-I, we set out to design and build a more efficient and easily operable implementation of our basic architecture. The result of this effort was VICE-II, our current file system.

VICE-II. In VICE-II, a single process on each cluster server services all file server requests from clients to that cluster server. This process uses a *lightweight process package (LWP)* with nonpreemptible scheduling to concurrently service many client requests. The RPC package is integrated with the LWP, thereby allowing the file server to be concurrently making or servicing one RPC per lightweight process. The RPC package is built on top of a low-level datagram abstraction and subsumes the demultiplexing, retransmission, and low-level failure detection functions that were provided by TCP in VICE-I. There is an RPC connection per client, but there is no a priori binding of lightweight processes to these connections. Instead, a pool of lightweight processes service client requests on all connections.

The use of a single server process makes it possible for us to maintain virtual-memory caches of many data structures that were kept in the file system in VICE-I. This improves performance and avoids the resource limitation problems, excessive paging, and context switching we encountered in VICE-I. The RPC package places no practical bounds on the number of clients who can be simultaneously connected to a server; each connection uses a small amount of virtual memory for state information, but no other resource.

In VICE-II, we use the UNIX file system on servers only to provide access to disk blocks, to manage storage allocation for files, and to maintain in-memory buffers of recently used disk blocks. The VICE directory structure is built on top of this low-level interface and does not appear as a UNIX directory structure on the server. We believe this will provide us with much greater efficiency in accessing files.

We have introduced the notion of a *volume* as the basic abstraction for administrative and operational purposes. A volume is a collection of VICE files comprising a partial subtree of the file system hierarchy and is typically quite small; each user in our system currently has a volume allocated to him or her. Tape backup and restoration, application of disk space quotas, and read-only replication are all done on individual volumes. The root of a volume may be arbitrarily relocated in the VICE file hierarchy, and in this respect, volumes resemble mountable disk packs in a conventional file system. Volumes are visible only at the VICE–Venus interface and are transparent to users and application programs. File location information is now obtained from a *volume location database*, replicated at all servers. When a Venus needs to locate a volume, it queries any server and caches the reply. The cached information is only treated as a hint, since volumes can be moved between servers.

The VICE–Venus interface in VICE-II uses unique *file identifiers* (fids) rather than full path names. A fid contains a volume number, a key into the volume index, and an additional field to ensure uniqueness within the volume. Fids remain invariant across renames and are therefore the key to making the renaming of directories possible. The translation of full path names into fids is done by Venus, which caches each directory encountered during translation. For robustness, modifications to directories can only be done by servers. Symbolic links are also interpreted by Venus.

Cache management is an area where VICE-II differs conceptually from VICE-I. In VICE-II, Venus may request a server to maintain a *callback* when it fetches a file or directory. If the file or directory is ever modified by anyone else, the server will inform each Venus with a callback on it that its cache entry has been invalidated. Venus can use cache entries with callbacks on them without any further validation, thereby cutting down significantly on client–server traffic. Servers are free to break callbacks at any time, even if the corresponding files are unchanged. This may happen, for instance, if a server discovers that it is expending too much memory or computational resources in maintaining callback state. Clients will revalidate cache entries as in VICE-I. Caches are still write through in VICE-II, but the cache replacement algorithm is based on the total space used by cached files.

VICE-II Status. VICE-II has been in use by the ITC for about five months and is still in the process of being debugged. Some of the functions, such as authentication, are still being integrated into the system. Nevertheless, even our limited experience with this system confirms its superiority to VICE-I. The ability to have symbolic links in VICE and to rename directories has enhanced the usability of the system. The callback mechanism and the use of a single UNIX process per server have resulted in marked performance improvement. We have not encountered any UNIX resource limitation problems with VICE-II, even though we have about 45 workstations connected to each of two cluster

servers. The internal structure of the server, RPC, and Venus, however, is considerably more complex than in VICE-I and has made debugging more difficult. A server process crash is now no longer a matter of a single user being inconvenienced.

Although much tuning, development, and refinement need to be done to VICE-II, we are confident that it represents a sound basis for the evolution of the Andrew file system.

THE USER INTERFACE

The goal of the ITC in workstation software was to design and develop tools that allow application developers to easily exploit the graphics capabilities of workstations. A secondary goal was to encourage the implementation of consistent application-specific user interfaces. This is particularly important for novices, who are often overwhelmed by the diversity of application-specific knowledge they need to effectively use the system. It was also our goal to explore a variety of interface paradigms and develop skills in implementing them.

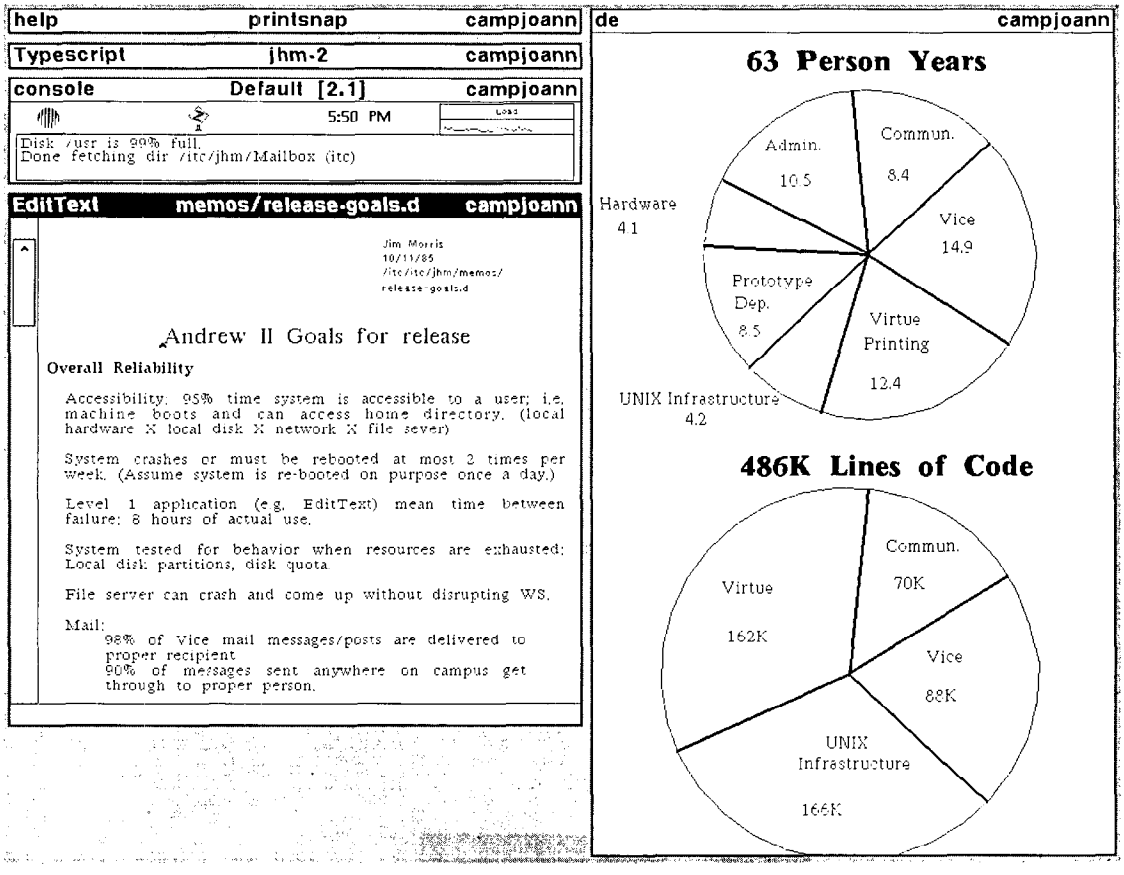
Our efforts have been directed toward three areas:

- a window manager that allows multiple processes to share a bit-mapped display,
- packages for manipulation of text and graphics, and
- applications using the window manager and the packages.

Window Manager

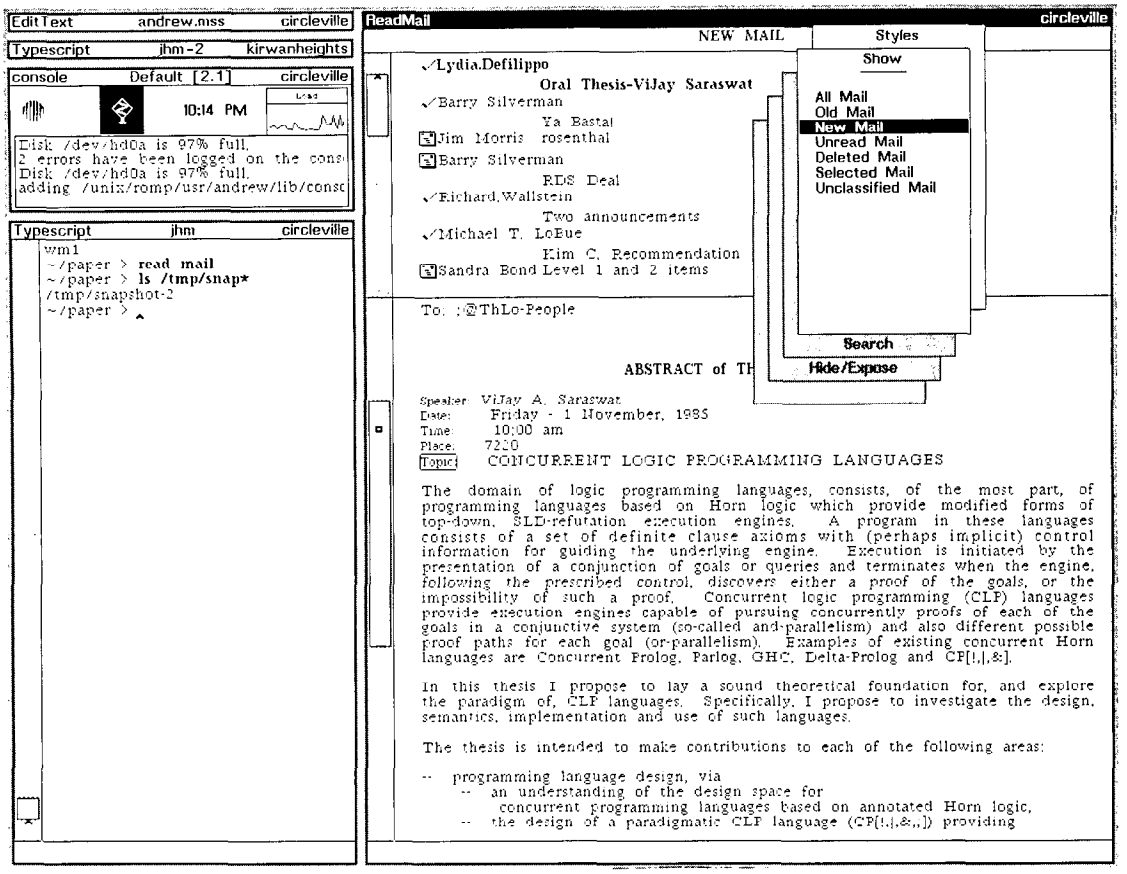
The building block for all applications is the window manager [8]. It virtualizes the display screen, dividing it into a number of rectangular areas whose size and shape are under the control of the user. Each window is attached to a process that can be oblivious to the presence of other windows and their processes, but that must be prepared to repaint its own window upon request.

A basic problem for a window manager is how to



The screen contains five windows. The top of each window is a title bar naming the application, its parameter, and the name of the workstation it is running on. The Help and Typescript windows have been shrunk to only their title bars. The console window displays various status items. A text editor and drawing editor consume most of the space.

FIGURE 7. A Screen with Several Windows



A set of pop-up menus appears in the mail-reading application. The various application-specific commands are grouped into five overlaid cards; sliding the mouse onto a portion of a card brings it to the top. The fourth card is currently on top, and the command for retrieving new mail has been selected.

FIGURE 8. Popped-Up Menus in a Mail-Reading Program

mediate between the user and the programs on how much of the screen is devoted to each program. Most window managers, for example, the Macintosh, follow the overlapping window approach in which the windows look like pieces of paper laid on top of each other, and the user selects the placement and size of each window. We chose a different approach, based partly on the Xerox Star [25] and Cedar [32] systems where the display screen is *tiled* with nonoverlapping windows. The user can adjust the boundary between windows, and windows can be completely hidden or quickly shrunk to only a title bar in order to free up space. Figure 7 shows a screen with several windows, some of which are only title bars. This scheme has two advantages: It is easy to program and requires the user to make fewer detailed decisions about arranging the screen.

The window manager multiplexes keyboard and mouse input for the various processes. Keystrokes are directed to the process whose window holds the mouse cursor; as feedback, that window always has a black title bar. The user can communicate with the process using pop-up menus; depressing a mouse button causes a set of process-specific commands to be displayed on small overlaid rectangles. After a command is selected, the menu disappears as it is executed. Figure 8 shows a screen with a set of menus popped up for the mail-reading program.

The window manager has been in use for two years, and its user interface has evolved considerably. The initial window layout policy completely filled the screen, and windows resized whenever the layout changed. A new window split the largest existing window; shrinking or hiding a window caused others to grow. This was both slow and confusing. We subsequently changed things to emulate the Cedar scheme: The screen is still tiled, but the primary division is a vertical boundary between two columns of windows, and gray space appears at the bottom of each column signifying unused space. Creating, destroying, or altering a window usually has no affect on other windows, only on the amount of gray space. The initial menu

system used hierarchical menus, like Interleaf's, in which selecting certain entries revealed submenus. This became unusable at depths greater than three. Now menus look like overlaid cards that the user can riffle through with a mouse movement. Shrinking and expanding windows were initially done through menus, but now a single click in the headline bar suffices. These changes are the result of extensive experience with alternative implementations. The trend has been to provide simpler and more predictable behavior along with tuning for common operations.

Although we have had doubts about the user interface presented by the window manager and will continue to improve it, we have been very satisfied with its basic architecture as seen by the programmer. Figure 9 shows the typical process structure on a workstation, with a number of client processes communicating with the window manager. When a window needs to be redrawn, the appropriate client process is informed via a software interrupt. It is the responsibility of that client process to query the window manager for the new window coordinates and size, and to make the necessary low-level calls to the window manager to accomplish the redrawing of the window. The client process chooses whether to scale, clip, or recompute the display in some other way.

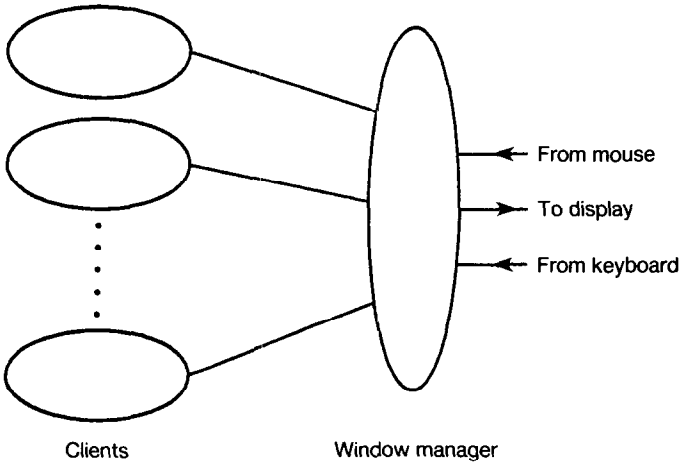
This approach was prompted by two circumstances: The workstations that Andrew would run upon were an ill-defined set, and our access to kernel sources for SUNs and other machines was difficult. The outcome is that the window manager can be run on any Berkeley UNIX system and is easy to adapt for particular display hardware—it has been demonstrated on three different machines and seven different displays, including color. People have been able to port it to a new machine with virtually no communication with the original authors. Furthermore, the use of sockets for communication makes it very simple to run applications on multiple machines while keeping their windows on one. Although this scheme prevents the application programs from getting at the full power of some displays, the performance of most interactive programs is surprisingly good. It can paint multifont text on a SUN 120 at over 4000 characters per second. This is due to three factors:

- We chose a set of primitives that keeps the most intensive operations such as font management and pixel-level character placement inside the window manager.
- We chose pure output operations that need no response from the window manager and are batched before being transmitted via the socket.
- The programmers of the window manager and certain applications worked very intensely.

The programmers' interface is simple to understand and has about 70 different procedures to perform various functions:

- *Window control* operations create and delete windows, change the contents of header lines, and request the current size of a window.
- *Drawing primitives* draw straight and curved lines and create filled regions. Rasterops on selected rectangles of the screen can be performed. Pixel coordinates are used to specify position. The set of graphics primitives is not quite as rich as that found on the Macintosh or in Postscript. We have implemented support for color, but have not used it extensively.
- *Text primitives* allow the display of a string at any pixel position in any font. The client names the font, and the window manager attempts to match it on a best-efforts basis. A good font representation has been designed to support performance.
- *Input* operations enable and disable keyboard input, mouse events in which the process is interested, and the shape of the cursor. Characters can also be output to a screen *cut buffer* or input from it.
- *Menu* operations allow the client to dynamically define the contents of menus—the contents of the menus and the actions taken on their selection are client specific. The client process specifies the items that are to be in the menu and the character sequence that is to be sent to it if that menu item is selected by the user.
- *Multiple windows* are supported by a set of operations to select input from and direct output to a particular window.

The window manager's procedures can be invoked from four different programming languages: C, Pascal, Fortran, and Lisp. Many applications and packages have been written on top of the window manager including a GKS (Graphical Kernel System) package.



The window manager controls the screen, keyboard, and mouse. It is a user process requiring no operating system privileges and has been carefully written to isolate display dependencies. Each application is also a process and talks to the window manager over a *socket*, a feature of the Berkeley UNIX 4.2 release that allows processes to communicate with streams of bytes.

FIGURE 9. The Window Manager Process Structure

Packages

Built on top of the window manager is a collection of data types called the *base editor tool kit.* In addition to a *programming interface*, each data type also possesses a well-defined *user interface*: a set of operations that a user can perform using mouse or keyboard input.

Application programs that use the tool kit exclusively for their interactions with users are benefited in a number of ways:

- The tool-kit interface is a higher level interface than the window manager and relieves the application developer from the many details associated with the display of justified, multifont text.
- The user interfaces of programs that use the tool kit are more likely to be mutually consistent than those of programs with independently developed user interfaces.
- It is easier to exploit graphics hardware and to obtain good performance by carefully tuning the implementations of a small number of data types than by refining a larger number of individual application programs.

The most basic data type in the tool kit is a *view*, which corresponds to a rectangular screen region within which an instantiation of another data type may be displayed. The latter may be a primitive data type or a composition of data types. A *document* is a data type that may be used whenever text manipulation of any kind is involved. Documents may range in size from a short label to an entire file. A view of a document is essentially a focus of interest on that document. Regions of text within a document may be demarcated with *markers* whose specific semantics depend on the application program. A *scroll bar* is a data type used in conjunction with a view of a document and is used to make different parts of the document visible on the screen. The tool kit includes a family of data types referred to as *buttons.* These are labeled, rectangular screen objects, each of which is associated with a set of procedures to be called when a specific event, such as a mouse click, occurs. Individual members of this family are used to represent scalar data types such as Booleans, finite sets, and strings.

The tool kit incorporates a *layout* mechanism, which deals with the physical placement of instantiations of data types within a window. Using high-level hints and placement constraints supplied by the application program, this mechanism uses heuristics to determine the actual sizes and locations of individual items within a window. When a window is moved or reshaped by the user, the layout mechanism is responsible for appropriately reconfiguring and redrawing that window.

A second large package, called Grits, supplies personal database services. A database consists of an arbitrary number of records, each of which can contain an arbitrary set of fields of any size. This flexibility makes Grits ideal for dealing with relatively unstructured information. Given the basic corpus of data, one can construct ordered indexes into it. There is a simple query language available in both library and interactive forms. The program library supports a set of layouts especially tailored to display individual records and indexes. Grits does not address the problems of locking records in VICE; related files must be locked in order to perform updates.

Applications

Many applications that use the capabilities of a bit-mapped display have been developed using the window manager and various packages. Many of the early versions were developed by the ITC, but recently, other groups at C-MU have assumed a major role in application development.

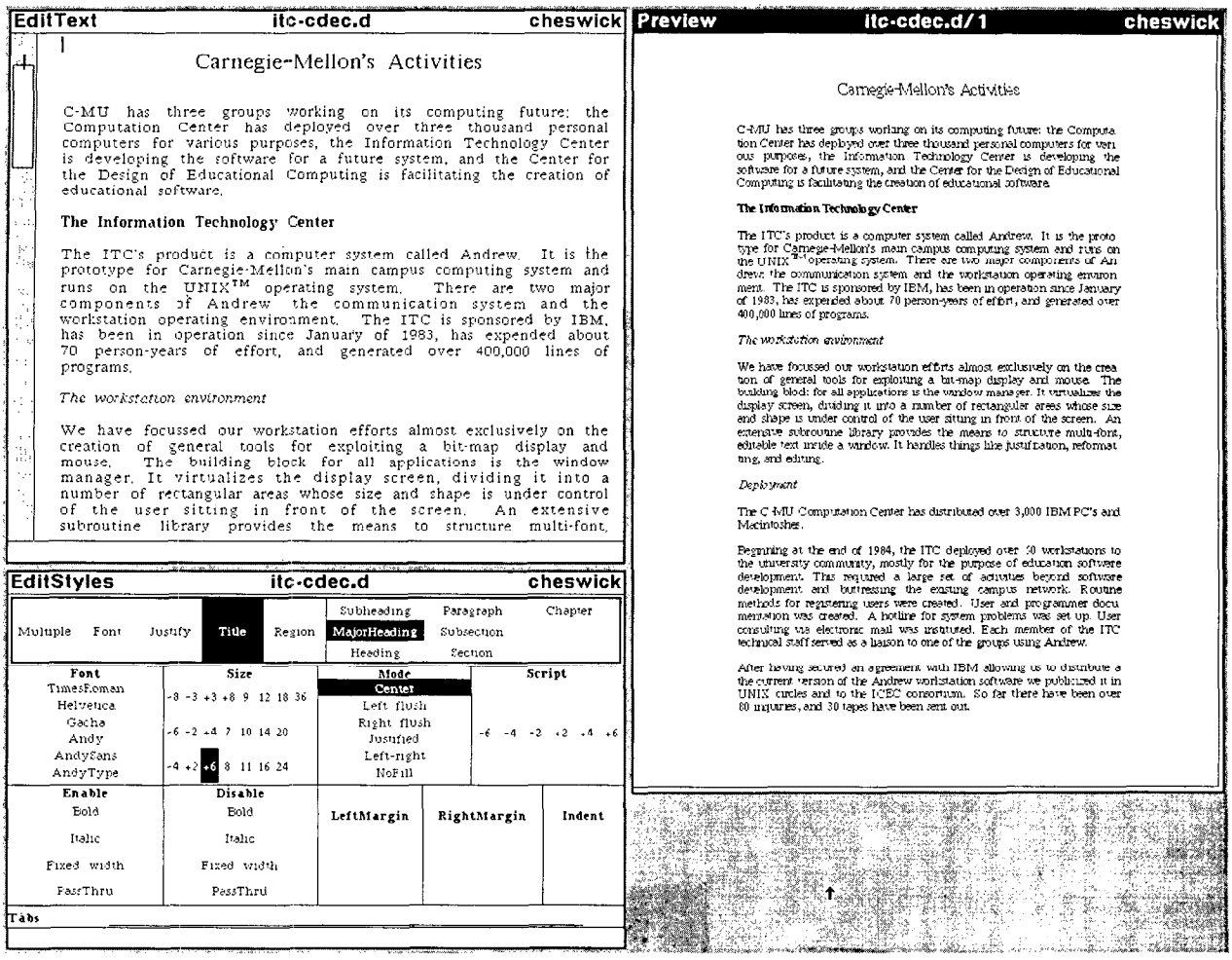
A text editor with dynamic formatting capability is the most popular application in general use. This editor is superficially similar to a "what-you-see-is-what-you-get" (WYSIWYG) editor, but differs from the latter in that it makes no attempt to produce a replica of a printed page. Rather, it attempts to format the text as best it can for the screen—reshaping an editor window automatically reformats the text to fit the new window. The appearance of the text is controlled by *style* directives in the document representation. Normally, these directives are invisible, except for their effect—for example, making some text bold—but the editor can be put into a mode that makes all styles explicit in a Scribe-like notation [19]. The styles in a document can be generic—a typical style might be "Major Heading." The actual appearance of styles can be controlled through a style editor that allows the user to specify, for example, that a major heading is centered, six points larger than its surrounding context, and bold. Figure 10 (p. 198) shows a document under the control of a style editor. The document compilers Scribe and Tex are also available on Andrew workstations. So far, we have used four different printers to print documents created on the system.

The standard Teletype driver was replaced by a typescript application supported by the base editor. This allows the user access to all normal text editing operations such as scrolling, cutting, and pasting when giving commands to the UNIX shell.

Several drawing editors that allow figures and drawings to be created interactively have been written or imported. They incorporate different paradigms and such features as constraint solving, automatic scaling, and animation. Some directory management applications have also been implemented illustrating several approaches: iconic symbols for files, directories as text files, and explicit pictures of trees. None has yet become comprehensive enough to replace the basic UNIX shell as the tool of choice.

Mail and bulletin-board browsing programs have been implemented using the base editor tool kit and the Grits database facilities. They allow the separation of mail into classes, scanning for particular subjects, etc.

An implementation of the Tutor programming language [24] has been nearly completed. Various versions of Tutor have been used at the University of Illinois and elsewhere to produce many hours of instructional material. This latest version, called C-MU Tutor, ex-



The EditText window shows an editable document, itc-cdec.d, whose generic styles are specified using the EditStyles tool. On the right, a Preview window shows the same document as it will appear when printed.

FIGURE 10. The Text Editor, Style Editor, and Preview

ploits the facilities of Andrew to create an interactive graphics-oriented programming environment. Figure 11 shows a Tutor program and the picture it has created. C-MU Tutor supports a rudimentary form of programming by example: The user can alter the picture in certain ways and have the program adjusted to produce the new picture.

EXPERIENCE AND PLANS

Although Andrew was (and is) far from completion, we deployed it to a small group at C-MU starting in December 1984. Over 50 SUN workstations were made available to people who had an interest in producing educational software. The workstations are spread rather uniformly over the campus, appearing in all six colleges and virtually every academic building. Currently there are over 500 registered users.

A survey of the user community in the summer of 1985 revealed that users liked the system, and for many, simply having a personal workstation running a full UNIX system is the most important thing. The piece of software most appreciated is the text editor. The most frequent complaints are that the system is unreliable and runs too slowly. The former is primarily due to disk-related hardware failures and resource exhaustion that neither the software nor the human support staff is yet equipped to deal with.

The performance problems are the sort to be expected of a new system: All the parts still are a little too slow, and people overuse them in a general spirit of exploration. For example, relatively mundane utility programs that probably should be simple shell commands appear as elaborate control panels using several fonts. The trivial act of resizing the window containing such a control panel can sometimes bring the system to its knees: The base editor library recomputes the layout for the window, changing the actual sizes of several fonts; then the window manager must fetch the fonts

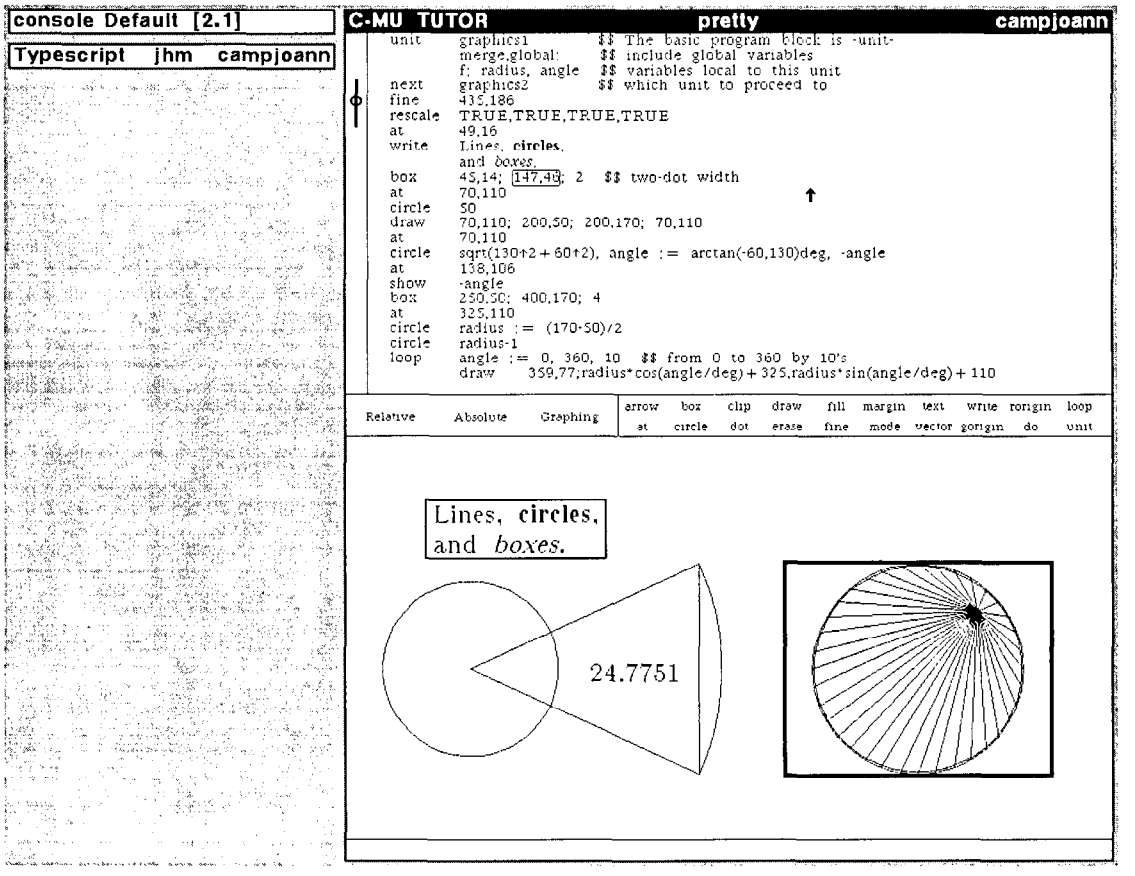
from VICE in order to repaint the window. Although the implementors might enjoy witnessing all this activity—and are amazed that it works at all—most users simply get impatient.

Aside from learning hundreds of ways in which Andrew needed improvement, we also learned that maintaining a distributed computing system is a formidable task. Even though our user population is small, it is widely distributed. Tools for troubleshooting are badly needed, and a sizable staff is required.

Despite the preliminary nature of the system, many faculty members have created very interesting applications in several areas: nonlinear differential equations, building design, chemical equilibrium analysis, American history, circuit design, scholarly writing, circuit analysis, and music synthesis. In addition, in February 1984, we began to distribute VIRTUE to other institutions. Over 40 sites have received the source code, under license for experimental use and assessment of its facilities.

Andrew is currently at the midpoint of its expected development period. The major past and desired milestones are summarized in Table I (p. 200). There are many other components of Andrew we have begun to work on or are contemplating:

- It is being ported to other workstations.
- The mail and bulletin-board systems are being overhauled and extended.
- Access to VICE over slow communication media is needed to support remote use.
- Supporting non-VIRTUE workstations, especially IBM PCs and Apple Macintoshes, is planned.
- A new version of the editor to support text, graphics, tables, and equations is under way.
- The VICE file system must have an archival subsystem.
- Support for more printers is needed.
- We shall import some key commercial applications including a spread sheet and a database package.
- We shall move most workstations to the IBM token ring.



The Tutor program in the top of the window has drawn the picture in the lower part of the window. The typefaces in the picture are controlled simply by attaching styles to the corresponding characters in the program. If the user clicks the mouse on some point in the picture, the coordinates of that point will replace the selected numbers in the program (147, 46) and thereby move the lower right corner of the box surrounding the text.

FIGURE 11. Tutor Figure

CONCLUSION

As mentioned at the beginning of this article, the ITC was created to design and implement a computing environment to serve as a unifying presence in the educational, administrative, and social life of C-MU. To meet this challenge, a system representing a synthesis of personal computing and time-sharing has been designed. The nature of the problem has necessitated the use of state-of-the-art techniques in LAN technology, distributed file system design, and user interface design. Using existing hardware, a prototype has been implemented with a view toward testing our ideas. The experience to date indicates that the design is fundamentally sound, though refinements are necessary in a number of areas. As Andrew grows, there will inevitably be many iterations over the design and implementation of various parts of the system.

It is appropriate to ask what is unique and noteworthy about the project. The most fascinating aspect is its scale and diversity of application. Never before has there been an attempt to support so many autonomous computers, each under the control of an unconstrained, untrained individual. Reliability, performance, and usability requirements conspire to make the design of such a system an intellectual challenge of the first magnitude. There are several specific areas where we feel we are advancing the state of the art:

- *Machine-independent raster graphics.* The design of the window manager has allowed us to run the workstation software on three different machine architectures and several different displays. Porting the system to a new display can often be done in less than a day.

TABLE I. Major Milestones in the ITC Project

Oct. 1982	IBM–C-MU contract signed, establishing the ITC.
Jan. 1983	Project starts.
July 1983	Most hiring, specific goal definitions, and overall architecture complete.
Aug. 1983	Development system obtained.
Nov. 1983	First release of window manager in use.
Jan. 1984	First release of base editor tool kit available.
Mar. 1984	First application program using base editor tool kit available.
July 1984	File system prototype available for use.
Nov. 1984	File system redesign begins.
Dec. 1984	Prototype deployment on the C-MU campus begins.
Feb. 1985	Andrew distribution to other campuses begins.
Mar. 1985	100 workstations in use.
July 1985	400 registered users.
Sept. 1985	Demonstration of faculty-created applications.
Oct. 1985	Redesigned file system in use by ITC.
Dec. 1985	Deployment of improved Andrew; 200 workstations in use.
Sept. 1986	Significant student access; 400 workstations in use.
Dec. 1986	Campus recabling complete; token ring in use.

- *Large, secure, distributed file systems.* As we have discussed, the VICE file system provides a file service whose size-functionality product will exceed any other we know of. The size coupled with the security constraints has, however, imposed many new problems.
- *Ubiquitous, high-performance text editing.* Our multifont interactive text editor compares favorably with the best commercially available ones. It does everything a WYSIWYG editor can be expected to, and very quickly. Its unique attribute, however, is that it is available as a library and permits virtually all text handling in the system to use all its features. Mail systems, interactive programming languages, and many education applications have used it.
- *Mail and bulletin-board systems.* Because we see today's electronic mail and bulletin boards as the forerunners of a very comprehensive campus communication system, we have begun to implement them in a very general way. At the same time, we must cope with the bewildering diversity of electronic communication in the larger world.

In retrospect, it is obvious that a project of this scope cannot be completed in the nominal 150 person-years planned for it. Nevertheless, we have not narrowed the scope, in the belief that an exciting and promising prototype, however flawed, will somehow capture the support needed to bring it to maturity.

Credits and Acknowledgments. The work described in this article represents the creative efforts of the entire staff of the ITC over the past three years. This article was written by James Morris and Mahadev Satyanarayanan with help from James Peterson; the other coauthors played significant technical management roles. Here is a functional summary of contributions to the system:

- UNIX system support: Robert Cosgrove, David Rosenthal, Mike Kazar, Carolyn Councill, and Bob Sidebotham;
- Release management and tools: James Peterson;
- Deployment support: Barry Silverman, Lynn Brown, and Chris Thyberg;
- Window manager: James Gosling, Bruce Lucas, and David Rosenthal;
- Text editor and tool kit: James Gosling, Fred Hansen, and Andrew Palay;
- Graphic design: Dan Boyarski;
- User interface testing: Chris Haas and Sandra Bond;
- Ethernet internetwork: John Leong;
- Token ring development: Don Smith and Bryan Striemer;
- SNA development: Jon Rosenberg and John Drake;
- Grits database: Tom Peters;
- Mail and bulletin boards: Tom Peters, Bob Cosgrove, Jon Rosenberg, Nathaniel Borenstein, and Craig Everhart;
- Printing: Andrew Palay, Mike Conner, and James Peterson;

- Graphical editors: Marc Donner, Bruce Lucas, Tom Peters, and Andrew Appel;
- Directory managers: Fred Hansen, David Nichols, David Rosenthal, and Tom Peters;
- Distributed file system design: John Howard, Mike West, Mahadev Satyanarayanan, David Nichols, Bob Sidebotham, Mike Kazar, and Al Spector;
- File server implementation: Mike West;
- Workstation file manager: Dave Nichols and Mike Kazar;
- RPC: Mahadev Satyanarayanan and Jon Rosenberg;
- Volume structure: Bob Sidebotham;
- Hardware support: Bryan Striemer, Paul Crumley, Mark Lorence, Jack Hutchings, and Kris Hutchings;
- IBM PC development: Larry Raper;
- Documentation: Sandra Bond, Carol Janik, Chris Neuwirth, Diane Langston, and Margot Critchfield;
- General administration: Barry Silverman, Nancy Rosenthal, Susan Straub, Bob Staab, Michael LoBue, Susan Parker, and Michelle Langhorne.

Both C-MU and IBM deserve credit for their willingness to chart a course into unknown waters, and for providing an excellent working environment for the ITC. In particular, Douglas Van Houweling of C-MU and Keith Slack of IBM were immediately responsible for the creation of the ITC, and the setting of its initial directions. There are few universities that would commit their computing future to such an innovative system, and there is probably no other computer company that would provide so much support without an initial guarantee of payoff.

REFERENCES

1. Accetta, M., Robertson, G., Satyanarayanan, M., and Thompson, M. The design of a network-based central file system. Tech. Rep. CMU-CS-80-134, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, Pa., Aug. 1980.
2. Accetta, M. A network router. Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, Pa., Sept. 1983.
3. Balkovich, E., Lerman, S., and Parmelee, R.P. Computering in higher education: The Athena experience. *Commun. ACM 28*, 11 (Nov. 1985), 1214–1224.
4. Ball, J.E., Barbacci, M.R., Fahlman, S.E., Harbison, S.P., Hibbard, P.G., Rashid, R.F., Robertson, G.G., and Steele, G.L. The Spice Project. Computer Science Research Review, Carnegie-Mellon Univ., Pittsburgh, Pa., 1981.
5. Birrell, A.D., Levin, R., Needham, R.M., and Schroeder, M.D. Grapevine: An exercise in distributed computing. In *Proceedings of the 8th Symposium on Operating Systems Principles* (Asilomar, Calif., Dec.). 1981.
6. Brotz, D., and Levin, R. Laurel. In *Alto User's Handbook*, B. Lampson and E. Taft, Eds. Xerox Palo Alto Research Center, Calif., Sept. 1979.
7. Defense Advanced Research Projects Agency, Information Processing Techniques Office. *RFC 791: DARPA Internet Program Protocol Specification*. DARPA, Sept. 1981.
8. Gosling, J.A., and Rosenthal, D.S.H. A network window manager. In *Proceedings of the 1984 Uniforum Conference* (Washington, D.C., Jan.). 1984.
9. Gray, J.P., et al. Advanced program-to-program communications in SNA. *IBM Syst. J.* 22, 4 (Dec. 1983).
10. Hiltz, J., Starr, R., and Turrof, M. *The Network Nation*. Addison-Wesley, Reading, Mass., 1979.
11. Lampson, B., and Taft, E., Eds. *Alto User's Handbook*. Sept. 1979.
12. Leong, J. Data communication at CMU. Tech. Rep. CMU-ITC-85-043, Information Technology Center and Computer Center, Carnegie-Mellon Univ., Pittsburgh, Pa., July 1985.
13. Leong, J. Nuts-and-bolts guide to Ethernet installation and interconnection. *Data Commun. 14*, 10 (Sept. 1985).
14. Metcalfe, R.M., and Boggs, D.R. Ethernet: Distributed packet switching for local computer networks. Tech. Rep. CSL-75-7, Xerox Palo Alto Research Center, Calif., May 1975. (Reprinted Feb. 1980.)
15. Nelson, B.J. Remote procedure call. Ph.D. thesis, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, Pa., May 1981.
16. Ousterhout, J.K., Da Costa, H., Harrison, D., Kunze, J.A., Kupfer, M., and Thompson, J.G. A trace-driven analysis of the UNIX 4.2BSD file system. In *Proceedings of the 10th ACM Symposium on Operating System Principles*. 1985.
17. Popek, G., Walker, B., Chow, J., Edwards, D., Kline, C., Rudisin, G., and Thiel, G. LOCUS: A network transparent, high reliability distributed system. In *Proceedings of the 8th Symposium on Operating Systems Principles* (Asilomar, Calif., Dec.). 1981.
18. Rashid, R.F., and Robertson, G.G. Accent: A communication oriented network operating system kernel. In *Proceedings of the 8th Symposium on Operating Systems Principles* (Asilomar, Calif., Dec.). 1981.
19. Reid, B.K., and Walker, J.H. Scribe introductory user's manual. Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, Pa., May 1980.
20. Ritchie, D.M., and Thompson, K. The UNIX time-sharing system. *Bell Syst. Tech. J. 57*, 6 (July–Aug. 1978).
21. Satyanarayanan, M. A study of file sizes and functional lifetimes. In *Proceedings of the 8th Symposium on Operating Systems Principles* (Asilomar, Calif., Dec.). 1981.
22. Satyanarayanan, M. RPC user manual outline. Tech. Rep. CMU-ITC-84-011, Information Technology Center, Carnegie-Mellon Univ., Pittsburgh, Pa., 1984.
23. Satyanarayanan, M., Howard, J.H., Nichols, D.N., Sidebotham, R.N., Spector, A.Z., and West, M.J. The ITC distributed file system: Principles and design. In *Proceedings of the 10th ACM Symposium on Operating System Principles* (Dec.). 1985.
24. Sherwood, B.A., and Sherwood, J.N. *The MicroTutor Language*. Stipes, Champaign, Ill., 1985.
25. Smith, D.C., Irby, C., Kimball, R., and Harslem, E. The Star user interface: An overview. In *Proceedings of the National Computer Conference*. 1982.
26. Smith, S., and Sherwood, B.A. Educational uses of the PLATO computer system. *Science 192* (1976).
27. Sproull, L., and Kiesler, S. Reducing social context information: The effects of electronic mail on organizational communication. Dept. of Social Science, Carnegie-Mellon Univ., Pittsburgh, Pa., Oct. 1985.
28. Strole, N. A local communications network based on interconnected token-access rings: A tutorial. *IBM J. Res. Dev. 27*, 5 (Sept. 1983).
29. Svobodova, L. File servers for network-based distributed systems. *Comput. Surv. 16*, 4 (Dec. 1984), 353–398.
30. Technical Committee Computer Communications of the IEEE Computer Society. *IEEE Standards for Local Area Networks: Token Ring Access Method and Physical Layer Specifications (ANSI/IEEE Std 802.5-1985)*. The Institute of Electrical and Electrical Engineers, 1985.
31. The Task Force for the Future of Computing, A. Newell, Chairman. The future of computing at Carnegie-Mellon University. Available from authors.
32. Teitelman, W. A tour through Cedar. *IEEE Softw. 1*, 4 (Apr. 1984).
33. Thacker, C.P., McCreight, E.M., Lampson, B.W., Sproull, R.F., and Boggs, D.R. Alto: A personal computer. In *Computer Structures: Principles and Examples*, D.P. Siewiorek, C.G. Bell, and A.N. Newell, Eds. McGraw-Hill, New York, 1982.
34. West, M.J., Nichols, D., Howard, J.H., Satyanarayanan, M., and Sidebotham, R.N. The ITC distributed file system: Prototype and experience. Tech. Rep. CMU-ITC-040, Information Technology Center, Carnegie-Mellon Univ., Pittsburgh, Pa., 1985.

CR Categories and Subject Descriptors: C.0 **[General]**: *systems architecture*; C.2.1 **[Computer-Communication Networks]**: Network Architecture and Design; C.2.4 **[Computer-Communication Networks]**: Distributed Systems; C.2.5 **[Computer-Communication Networks]**: Local Networks; H.1.2 **[Models and Principles]**: User/Machine Systems; H.2.4 **[Database Management]**: Systems

General Terms: Design

Additional Key Words and Phrases: Andrew, Information Technology Center

Authors' Present Address: James H. Morris, Mahadev Satyanarayanan, Michael H. Conner, John H. Howard, David S.H. Rosenthal, and F. Donelson Smith, Information Technology Center, Carnegie-Mellon University, Schenley Park, Pittsburgh, PA 15213.