CMU-ITC-84-035

The ITC Project: An Experiment in Large-Scale Distributed Personal Computing[1]

M. Satyanarayanan
Information Technology Center
Carnegie-Mellon University

October 1984

Abstract

The Information Technology Center, a collabrative effort between International Business Machines Corporation and Carnegie-Mellon University, is currently in the process of designing and implementing what may be one of the largest distributed computing systems yet attempted. This paper traces the origins of the project, discusses design issues pertinent to it, and gives an overview of the current status of a prototype implementation. The design described here combines the ease of information sharing characteristic of timesharing systems with the richness of user-machine interaction typical of personal computing. The resulting synergesis yields a model that may well become the standard computing paradigm of the next decade.

1. The Mission

Carnegie-Mellon University (CMU) is a small, technically sophisticated institution located on a geographically compact campus. These attributes, in conjunction with rapidly advancing technology in the computer industry and an enlightened administration have given rise to a project to produce a campus-wide integrated personal computer environment.

In 1979 the Computer Science Department at CMU (CMU-CSD) carried out an extensive study to determine the characteristics of a computing environment appropriate for computer science researchers in the 1980's. Their report [24] described the specifications of a hypothetical high-function personal computer called the *Spice* machine. The key attributes of this machine were the use of a bit-mapped display and a pointing device, virtual memory, attachment to a local area network, a multi-process operating system, and a sophisticated user interface. Using off-the-shelf hardware that approximated these requirements, the Spice project at CMU-CSD has been engaged in research to design and build the software component of the Spice machine.

In September 1981, the president of CMU created a multi-disciplinary task force to consider the role of computing in CMU's educational program. The task force report [16] emphasised the need to maintain and

[1] A preliminary version was presented as an invited paper at the Networks 84 Conference held in October 1984 at Madras, India.

enhance CMU's preeminence in the area of computing. The proliferation of personal computers points to a situation where small, technical universities will have one computer per person on campus by 1990. The task force recommended that this process be controlled and accelerated at CMU, thereby giving it an opportunity to set the standard for educational computing in the next decade.

Experience with personal computing has been overwhelmingly positive in small groups such as individual departments and research laboratories. However, the cost/benefits of personal computing are not yet so demonstrably superior that businesses or large universities are willing to adopt it as their model of computing. With its particular strengths, CMU is in an excellent position to conduct an experiment in large-scale personal computing. CMU students would thus not merely be computer literate, but would actually advance the use of technology in the world.

The task force report strongly emphasised the importance of communication between all the personal computers and mainframes on campus so that maximal sharing of information and resources would be possible. The natural outcome of such a strategy would be a network of personal computers which is larger than any developed hitherto.

Contemporaneously, IBM was developing a strategy for increased involvement with universities. Its primary goal was to become a leading purveyor of computing systems to research-oriented universities. Discussions with CMU resulted, and the advantages of a collaboration soon became obvious to both parties. CMU could be a showcase for advanced IBM products, and a source of ideas for future products. IBM's development community would also benefit from interaction with the university. For its part, CMU would receive considerable development resources and the confidence that the system it developed would not become a one-of-a-kind curiosity.

In October 1982, CMU and IBM signed a contract which created the Information Technology Center (ITC). Located on the CMU campus, the charter of this organisation is to design and develop a system based upon IBM hardware to support CMU's ambitious plans for educational computing. In a nutshell, the mission of the ITC is to provide the software underpinnings needed to harness the energy and creative talents of the entire university in making computers an integral part of the educational process.

2. The Computing Paradigm

2.1. Personal Computing

Personal computers have been touted by manufacturers and users as the wave of the future. Like other buzzwords, the phrase "personal computing" has come to acquire a variety of shades of meaning. What does it connote in the context of the ITC project?

The era of personal computing was ushered in by two independent events in the 1970's. The first of these was the introduction of stand-alone microprocessor-based computers by companies such as Apple Inc. The identifying characteristic of this class of machines was their cost: it had to be affordable to small businesses and individuals. This cost constraint and the available technology dictated the architecture, hardware implementation, and software aspects of these machines: floppy disks, 8-bit data paths, character-displays, and BASIC were the order of the day. From a user's point of view, the only real difference between using such a machine and using a timesharing system was the fact that the performance of the former was constant and predictable, unaffected by the activities of other users.

The second influence was the Alto project at Xerox's Palo Alto Research Center [27]. In a radical departure from traditional designs, this project sought to make possible a "paperless office," where electronic media could totally replace the printed page. A custom-built stand-alone machine was designed as part of the project. Integral to this design were 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 keystrokes were used by humans to communicate with the machine. The Alto project was thus as farreaching an advance over traditional timesharing user interfaces as timesharing was over batch processing. The model of man-machine communication first demonstrated in this project has now come to be accepted as a highly desirable one, particularly for novices.

The confluence of these two independent developments has yielded the class of personal computers envisioned in the ITC project: individual students and faculty will each possess a workstation with a bit-mapped display, pointing device, and user-interface software that exploits them. Within this general scenario there are, of course, many degrees of freedom in design. The challenge is to find a design that is friendly to a novice user, and yet does not hinder the expert. An important class of experts in this regard is the set of application developers who will create educational software.

The need to span a wide spectrum of user sophistication has motivated the ITC to require that the hardware possess a large address space, virtual memory, and a multi-process operating system, in addition to a bit-mapped display and mouse. The Sun Microsystems Inc. workstation, which possesses these features, has been chosen as the hardware for internal development in the ITC and pilot deployment to a small set of users. Hardware currently under development by IBM will be used in the large-scale deployment to students and faculty.

On the software front, the desire to have a shortened development cycle, a stable software base, and the ability to import existing applications has resulted in the adoption of Unix[2] as a de facto ITC standard. In the long run it is expected that workstations with a variety of operating systems will be integrated into this environment. Initially, however, they will all run 4.2BSD Unix with minor, upward-compatible ITC modifications.

2.2. Timesharing

An unanticipated side-effect of the popularity of timesharing was the use of computers as a vehicle for communication between users. Sharing of information via a file system, electronic mail, and bulletin boards 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 strictly computational capability.

In moving away from timesharing to personal computing, it would indeed be unfortunate if users lost the ability to communicate among themselves. The development of local area networks (LANs) specifically addresses this issue. Originating with the Ethernet [14] in the Alto project, the linking together of workstations by a LAN has become standard practice in institutional environments. Inter-machine mail and bulletin board facilities are typical utilities in a networked personal computer environment [4, 2]. Transmission of files between machines, and the use of a shared facility as a repository of files are also common [3, 26]. LANs also make possible the shared use of relatively expensive peripherals such as laser printers.

In a timesharing 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 is 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 login to read mail.

While a networked personal computing environment provides connectivity, it does not automatically imply

[2]In particular, the 4.2 Berkeley Standard Distribution (4.2BSD Unix) has been adopted.

the same case of sharing. Each workstation in such an environment has a unique network address which has to be specified when accessing files from it, or when sending mail to its owner. Further, explicit user actions are usually required to achieve sharing: before using a file, a user has to run a program to transfer it to the network node where he is currently located; changes made by him are not visible to other workstations until he transfers the modified file to them or to a central repository. A corollary to these observations is that user mobility is limited. One cannot create a file at one workstation, walk to another workstation, and access that file with the effortlessness that is possible when using geographically separated terminals of a timesharing system.

In a relatively small network, the limitations exposed in the previous paragraphs may not impact user productivity significantly. The ITC project, however, aspires to interconnect over 5000 workstations! Almost by definition, an academic environment requires a large amount of information sharing. A high degree of user mobility between dormitories, faculty offices, libraries, and laboratories is also essential.

In the light of these observations, the second fundamental goal of the ITC project becomes evident: to make possible an information sharing environment that has the same degree of ***Location Transparency*** and ***User Mobility*** that is available in a well-designed, modern, timesharing system. Unix, once again, is a good role model for this aspect of the system.

2.3. VICE and VIRTUE

The computing paradigm envisioned in the ITC is thus a marriage between personal computing and timesharing. It incorporates the flexibility and visually rich user-machine interface made possible by the former, with the the ease of communication and information sharing characteristic of the latter. This model is depicted in Figure 2-1.

The large amoeba-like structure in the middle, called VICE, is a collection of communication and computational resources serving as the information sharing 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 the overhead of a sophisticated user-machine interface.[3]

VICE provides a common name space for shared resources, particularly files. Users may thus access files in a uniform manner regardless of the specific workstations at which they are logged in, or at which the files were created originally. VICE has mail and bulletin board facilities, thus obviating the need for location-specific information in sending mail or posting notices. Software in each VIRTUE workstation makes these facilities of

[3]Rumour has it that VICE stands for "Vast Integrated Computing Environment." VIRTUE, of course, is one's only defense against VICE!

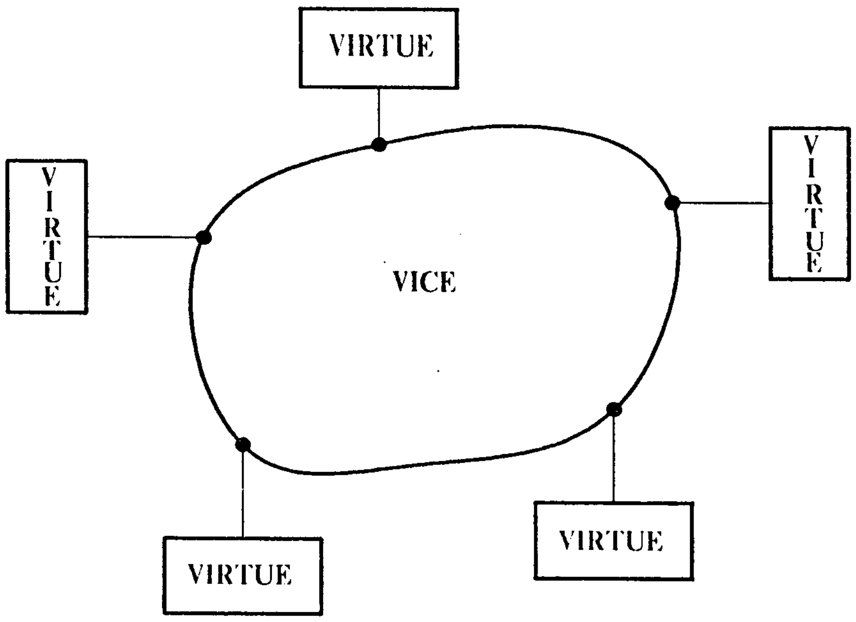


Figure 2-1: VICE and VIRTUE

VICE appear as a transparent extension of that workstation's operating system.

The VICE-VIRTUE interface has two important properties:

1. *It is a relatively static programming interface.*
   Enhancements to this interface will typically be made in an upward-compatible manner. This allows advances in technology to be taken advantage of, without system-wide 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 non-homogeneous workstations are attached to VICE, but share its resources in a uniform manner. Even at the present time two different types of machines (Sun Microsystems workstations and Digital Equipment Vaxes) running 4.2BSD Unix are capable of accessing VICE.

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 computers and the use of encryption on exposed parts of the network. No user programs are executed on any VICE machine. Barring Trojan horsemanship, therefore, VICE is an internally secure environment. The workstations, however, are owned by individuals who are free to modify the hardware and software in any way they choose to. Hence VIRTUE is not trusted by VICE, except for the duration of an authenticated session at the beginning of which credentials are exchanged, and during which communication between VICE and VIRTUE is encrypted.

The ITC project may be naturally decomposed into three parts:

1. *A Network Communication* component, dealing with the hardware and software needed for inter-machine communication.

2. *The Shared File System*, which is the first and most important shared resource in VICE.

3. *The User Interface* on an individual worktation, which is the aspect of the ITC project most visible to the average user.

We examine these topics in the next three sections of this paper.

3. Network Communication

3.1. Connection Structure

Conventional wisdom, based on LAN usage experience such as that reported in [6, 23], suggests that network utilization is rarely high enough to be a serious concern in designing a distributed system. However, the scale of the ITC project is one to two orders of magnitude larger than that of most existing networks. Consequently, it is possible that network utilization may become a serious source of performance degradation in a fully configured system. Even if network delays turn out to be inconsequential, there is still cause for concern: the shared computing elements within VICE may become bottlenecks. The evidence indicates that this is indeed a possibility in real systems [12].

The ITC project uses a design strategy that exploits locality of reference to reduce network and server utilization. Viewed at a finer granularity than Figure 2-1, VICE is composed of a collection of semi-autonomous ***Clusters*** connected together by a ***Backbone*** LAN. Figure 3-1 illustrates such an interconnection scheme, assuming a linear topology connection medium such as Ethernet. Each cluster consists of a collection of workstations and a representative of VICE called a ***Cluster Server***. Physical security considerations may dictate that cluster servers be co-located in small groups in machine rooms, even though each cluster server is logically associated with the workstations in its cluster. Being part of VICE, a cluster server only runs software trusted by the system administrators. There is no mechanism for users to run their programs on cluster servers.

The ***Bridges*** which 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. The key to effective use of this interconnection scheme for reducing network utilization lies in designing the software so that most network traffic remains within a cluster. Inter-cluster traffic via the backbone should be the exception rather than the rule. The file system described in Section 4 is designed with this goal in mind. There is conceptual similarity between this problem and that faced by sofware on Cm*, a multiprocessor with differential memory access times caused by the clustering of its constituent processors [11].

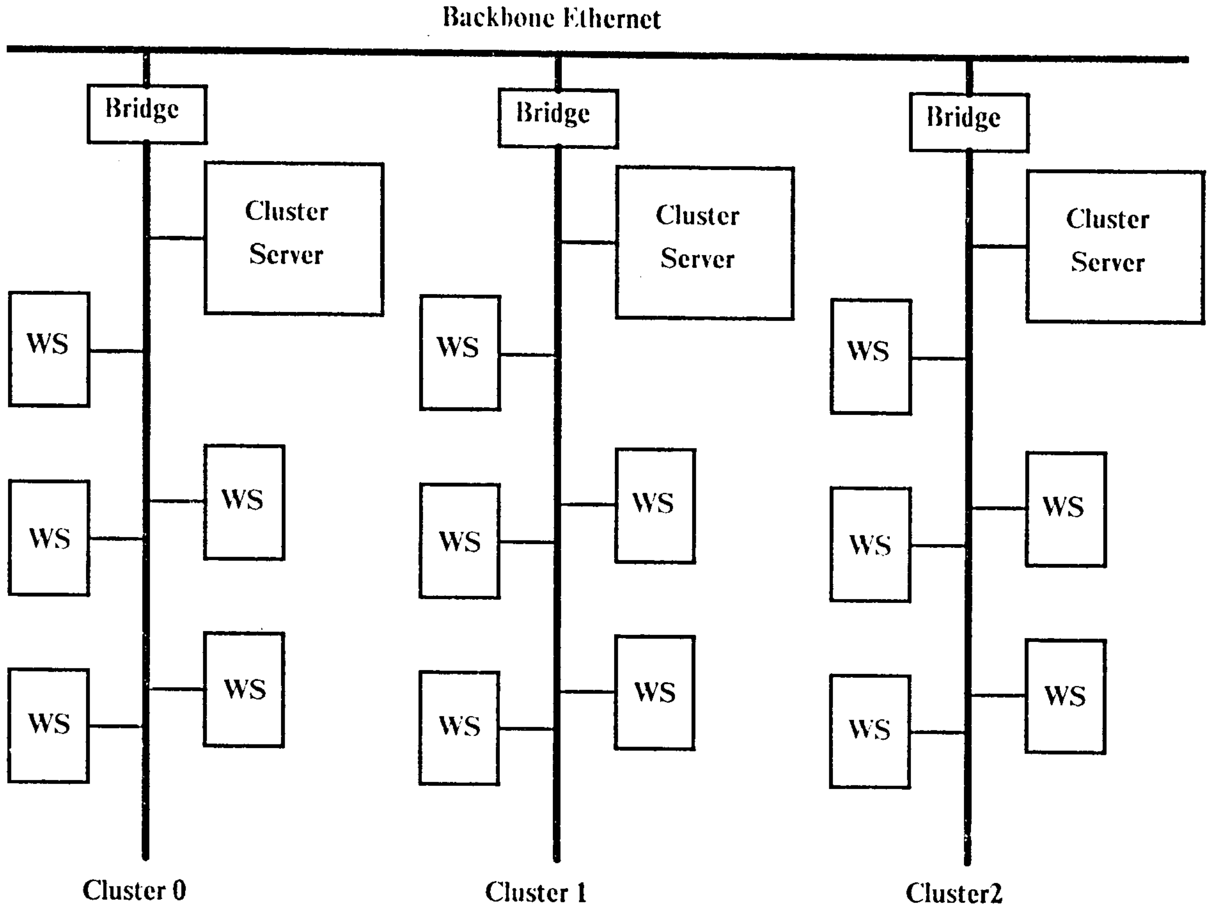


Figure 3-1: Ethernet VICE Topology

3.2. Network Hardware

Though the distributed nature of the ITC project is pervasive at all levels of the design, there is suprisingly little dependence on specific attributes of the network hardware. There is no requirement, for instance, that the connection media be capable of broadcast communication. The only features required of the network hardware are:

1. High speed (in the 1 to 10 Mbit/s range).

2. Low error rate (comparable to that of Ethernet).

3. Addressing and node attachment capability adequate for 5000 to 10000 nodes.

4. Availability as a standard vendor product, with a full range of accessories for maintenance.

5. Low-cost adaptors, to minimize the expense of VICE-VIRTUE attachment.

The long-range plans of the ITC call for the use of a token ring network currently under development by

IBM. This network is based on (but is not plug-compatible with) the token ring developed at IBM Zurich [5], and conforms to the IEEE Standard 802 [10]. In the short term, for development purposes and pilot deployment the ITC is using Ethernet as the networking medium.

There is some controversy over the relative merits of CSMA-CD technology (Ethernet in particular) versus token ring technology in a large-scale network. An eloquent defense of token rings has been put forth by Saltzer et al [19]. The compelling argument in favour of token rings is that workstations may be attached using wiring technology identical to that used in wiring telephones: one which has evolved over the years to allow ease of serviceability, fault detection, and fault isolation. The more ad hoc attachment scheme characteristic of Ethernet is convenient for small installations but may not be ideal for a user community as large as the entire CMU campus. Proponents of the Ethernet point to the fact that this is a mature technology, is in widespread use, and is supported by a large number of independent vendors.

It is not clear at this point in time how this controversy will be resolved. Usage experience with a pilot deployment using Ethernet and IBM sponsorship of the project are two factors that will undoubtedly play a role in the final decision. Fortunately, ITC-developed software is quite well insulated from the specifics of network hardware. Moving to a token ring network would require writing device drivers and a communication protocol package for the workstations and servers. The bulk of the file system and user interface software would be unaffected.

3.3. Network Protocols

Experience with networking over the last decade has emphasised the importance of using standardized communication protocols for inter-machine 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 the ITC, many of the departmental mainframes at CMU used the DARPA Internet Standard protocols [7] for communication. This is a family of protocols spanning the levels between the Application Layer and the Network Layer of the ISO Open System Interconnect Reference Model [8]. The family consists of an unreliable datagram protocol (IP/UDP) and a reliable byte stream protocol (IP/TCP) at the lowest levels, with end-user application protocols such as those for mail, file transfer, and remote login built on top.

Due to its sponsorship by the US Department of Defence, this protocol family has become the lingua franca of the Arpanet user community, of which CMU is a very active member. Further, the 4.2BSD Unix operating system, which is being used as a base for VICE and VIRTUE, already has implementations of these protocols built into it. Consequently, the ITC has adopted the Internet protocol family as its standard too.

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 at the ITC [21]. The distinctive features of this package are:

1. The transfer of bulk data objects, such as files, as side effects of remote procedure calls. This capability is used extensively in the file system for caching of files at workstations.

2. Built-in authentication facilities which allow two mutually suspicious parties to exchange credentials via a 3-phase encrypted handshake. This mechanism is used by servers in VICE to authenticate users.

3. Optional use of encryption for secure communication, using session keys generated during the authentication handshake.

4. The Shared File System

Network file systems have been the subject of investigation in a number of projects over the last few years [17, 1]. A good discussion of these designs may be found in Svobodova's survey [26]. Typically, these designs have been intended for networks with at most a few hundred nodes. With its ambitions to span an order of magnitude more nodes, the ITC distributed file system design problem had to be approached afresh: the designers had little confidence that an adaptation of an existing design would prove adequate to the task.

The goal of the ITC is to produce a distributed file system possessing the following properties:

Location transparency
A user need never know which network node a particular file is located at. For all intents and purposes, the distributed file system is viewed by users as a giant timesharing file system.

User mobility
A user can suspend work at one workstation, move to any other workstation and resume work without explicit actions to transfer files. A workstation is "personal" only in the sense that it is owned by a particular individual.

Security
It cannot be assumed that all users of the system are non-malicious. In particular, the hardware and software on workstations may be modified in arbitrary ways by their owners. Network communications cannot be assumed secure. The design of the system should allow easy and flexible sharing of resources (in particular, files) in such an environment.

Performance
The user-perceived performance should be no worse than that of a well-designed, lightly-loaded timesharing system. Users should not feel the need to make explicit file placement decisions on account of poor performance.

Expandibility
It should be possible to expand the system in a graceful and non-traumatic manner. It is expected that the total number of network nodes in the system will be between 5000 to 10000, thus making it one of the largest distributed file system in existence.

Reliability As users become dependent on this system, system non-availablity will become increasingly intolerable. While not as stringent as in a real-time process control environment, reliability and availability are none the less very important goals.

The following sections describe the main architectural features of a design that addresses these issues.

4.1. Naming

From the point of view of each workstation, the space of file names is partitioned into two subspaces: ***Local*** and ***Shared***. Figure 4-1 illustrates this partitioning of name spaces.

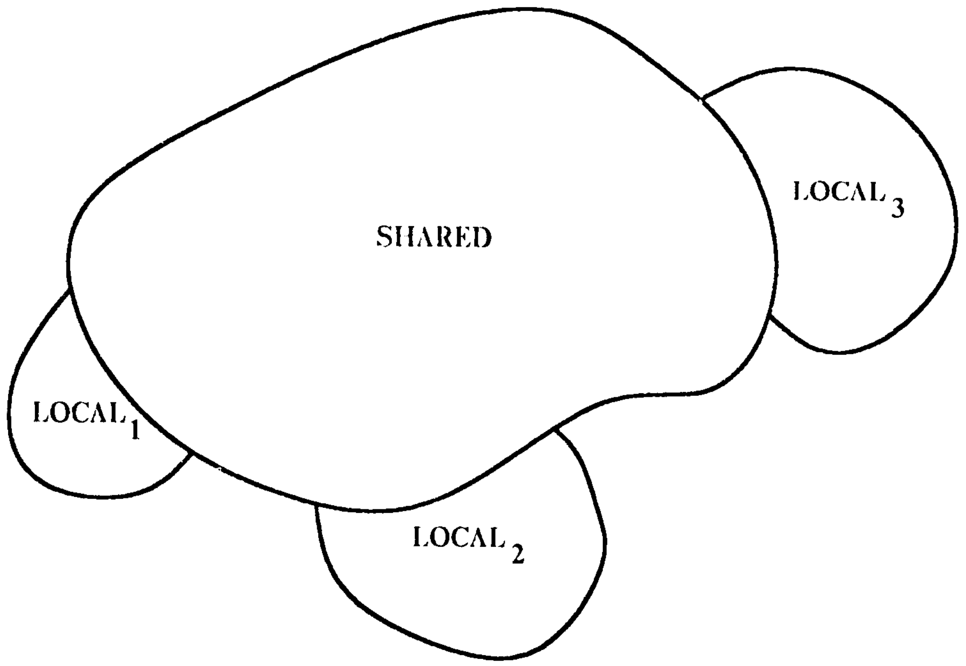


Figure 4-1: Shared and Local Name Spaces

The shared name space is the same for all workstations, and contains the majority of files accessed by users. The local name space is small, distinct for each workstation, and contains files which typically belong to one of the following groups:

1. System files essential for booting the workstation and for its functioning prior to connecting to VICE.

2. Temporary files, such as those containing intermediate output from compiler phases.

3. Data files which the owner of the workstation considers so sensitive that he is unwilling to entrust them to the security mechanisms of VICE. Such files cannot, of course, be accessed in a location transparent manner.

In addition, to improve performance and to allow at least a modicum of usability when disconnected from the network, certain commonly used system programs may be replicated in the local name space of each workstation.

Both the local and shared name spaces are hierarchically structured, and are similar to a timesharing Unix file system [18]. 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 "/vice" during workstation initialization. Figure 4-2 depicts this situation. Since all shared file names generated on the workstation have "/vice" as the pathname prefix, it is trivial to disambiguate between local and shared files.

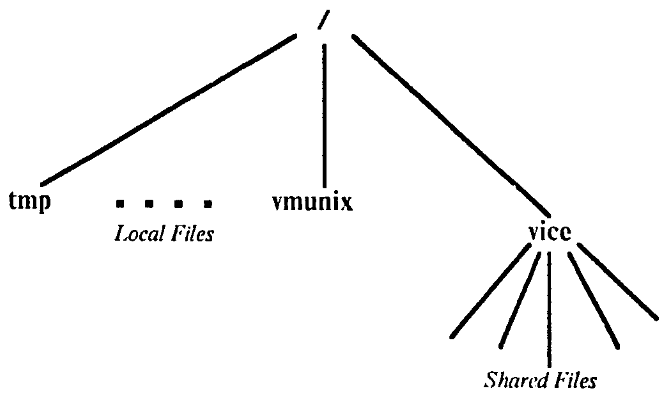


Figure 4-2: A Workstation's View of the File System

4.2. Data Distribution and Replication

Each cluster server in VICE runs a file server process which supports operations such as storing and retrieving files on that cluster server in response to workstation requests. The hierarchical file name space is partitioned into disjoint subtrees, and each such subtree is served by a single cluster server, called its ***Primary Custodian.*** Storage for a file, as well as the servicing of requests for it, are the responsibility of the corresponding primary custodian. Certain subtrees which contain frequently read, but rarely modified files, may have read-only replicas at other cluster servers. These read-only sites are referred to as ***Secondary Custodians*** for the subtree, and each of them can satisfy read requests independently. Write requests to that subtree have to be directed to the primary custodian of the subtree, which will perform the necessary coordination of secondary custodians. Changing the custodian of a subtree is a relatively heavyweight process: the design is predicated on the assumption that such changes do not occur on a minute-to-minute basis, but are batched together and typically done once a day.

Each cluster server logically contains a location database which may be queried by workstations to ascertain the custodian of any file. The query may be addressed to any cluster server, but performance considerations suggest that the communicating parties be on the same cluster. 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, the location database need only have an entry for the root of the subtree.

The location database changes relatively slowly for two reasons:

1. Most file creation and deletion activity occurs at depths of the naming tree far below that at which the assignment of custodians is done.

2. Reassignment of custodians is infrequent and is initiated via administrative procedures. Consequently, a specialized slow-update procedure which updates the custodianship information at all cluster servers is possible.

The assignment of custodians to the file subtrees of individual users is done so that there is a high probability of a user being at a workstation on the same cluster as his custodian cluster server. A faculty member, for instance, would be assigned a custodian on the same cluster server as the workstation in his office. This assignment does not affect the faculty member's mobility, since he could access his files from any other cluster on campus, albeit with some performance penalty.

4.3. Caching

In this design, caching of entire files on workstations is the primary mechanism used to attain location transparency, performance, and application code compatibility. Each workstation has a local disk, part of which is used to store files in the local name space, the rest being used as a cache of files in the shared name space.

When an application program on a workstation makes a system call to open a file, the request is first examined by the kernel 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 cache is checked for the presence of valid copy. If such a copy exists, the open request is treated as a local file 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 custodian. 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 a 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; this copy is then transmitted to the appropriate custodian. The cache thus behaves as a write-through cache on closes.

Checking the validity of a cached copy of a file is done by workstations rather than by a broadcast invalidation mechanism as in multiprocessor systems. This is because only a small fraction of the total update activity in the entire system is likely to affect an individual workstation. Broadcasting all such changes is likely to have a drastic effect on performance.

The caching mechanism allows complete mobility of users. If a user places all his files in the shared name space, his workstation becomes "personal" only in the sense that it is owned by him. The user can move to any other workstation attached to VICE and use it exactly as he would use his own workstation. The only observable differences would be an initial performance penalty as the cache on the new workstation is filled with the user's working set of files and a smaller, perhaps unnoticeable, performance penalty as inter-cluster cache validity checks, lock requests, and cache write-throughs are made.

The caching of entire files, rather than individual pages, also has a beneficial effect on performance:

1. Network overheads are minimized because custodians are contacted only on file opens and closes, and not on individual reads and writes.

2. The servers do not have to maintain state information about open files in the system.

Inevitably, there are some files which are far too large to fit in workstation caches. These are typically databases, such as the on-line card catalogue 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. Studies of file usage patterns in real systems have, in fact, shown that most files tend to be small [20, 12].

4.4. Other Issues

VICE provides multi-reader/single-writer synchronization for files. Read locks are optional, but a write lock prior to creating or overwriting a file is mandatory. These are more stringent requirements than those in Unix, but were felt to be necessary in view of the size of the shared file space, and the amount of concurrent usage activity on it. Application programs do not, of course, have to do explicit locking; it is done on their behalf by the caching software on their workstations. If communication with a workstation is lost, VICE automatically breaks all locks granted to that workstation.

In order to allow flexible yet controlled sharing of resources in VICE, an access list package has been developed. The protected objects in the file system are directories, not individual files. Typical modes of sharing between users lead us to believe that the overheads involved in associating an access list with each file is unwarranted. Entries in an access list may correspond to ***Users*** or ***Groups***. Users are typically humans, while groups are collections of users and other groups. The recursive structure of groups implies that a user may be a direct member of a group, or an indirect member via transitivity of the membership property. When accessing an object in VICE, the rights accrued to a user are the union of the rights possessed by all the groups that he is a direct or indirect member of. More details on the protection mechanism may be found in the design document [22].

For reliability, a directory can have a ***Backup*** custodian specified. This custodian contains an exact image of the original directory, and changes to the latter are asynchronously reflected to the backup. Currently, the backup copies of files are read-only by workstations. Algorithms for allowing workstations to directly modify backup directories are under consideration.

Further details on the design of the shared file system may be found in the design documents [9, 28].

5. The User Interface

In the user interface domain, the role of the ITC is to design and develop software that allows application developers to easily exploit the graphics capabilities of workstations. Since user-interface code in the past has tended to be highly hardware-specific, one of the goals of the ITC has been to develop software to insulate application developers from the details of the graphics hardware. Software has also been developed to simplify and encourage the implementation of consistent application-specific user interfaces. This is particularly important for novices, since they are often overwhelmed by the diversity of application-specific knowledge they need to possess to effectively use the system.

The user interface offerings of the ITC can be categorized as follows:

- A ***Window Manager*** that allows multiple processes to share a bit-mapped display.
- A ***Tool Kit*** of graphical data types for application developers.
- ***Applications*** built using the window manager and the base editor.

We examine each of these components in the following sections.

5.1. Window Manager

The window manager may be viewed as a Hoare-style monitor that manages the display, mouse, and keyboard attached to a workstation, allowing multiple processes to use these devices without interfering with each other. Two features of 4.2BSD Unix are essential to the window manager implementation: interprocess communication between ancestrally unrelated processes, and the ability to simultaneously wait for input from many different devices. No kernel modifications are necessary, since the window manager runs as a user process.

The screen is partitioned into non-overlapping rectangular ***Windows***, each associated with a unique ***Client*** process. A client can have more than one window associated with it, though this is not usually the case.

Figure 5-1 shows the typical process structure on a workstation, with a number of client processes communicating with the window manager. A specialized remote procedure call mechanism is used for communication between clients and the window manager. To obtain satisfactory interactive performance, remote procedure calls which do not return values are batched together prior to transmission.

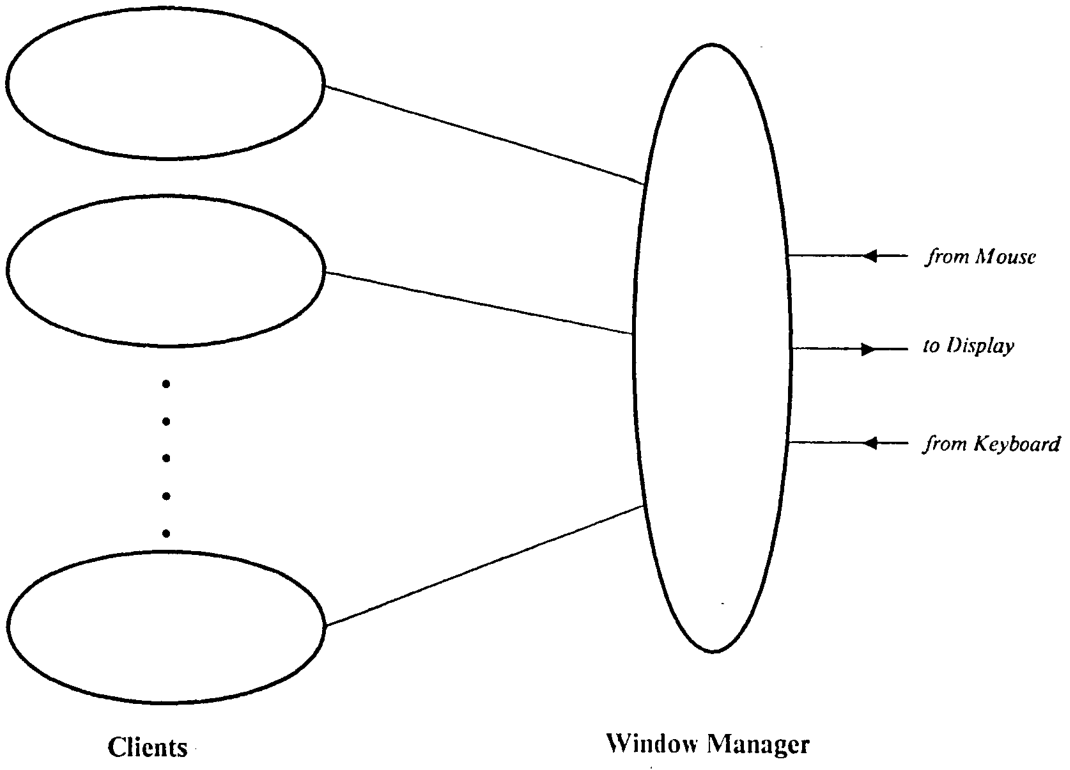
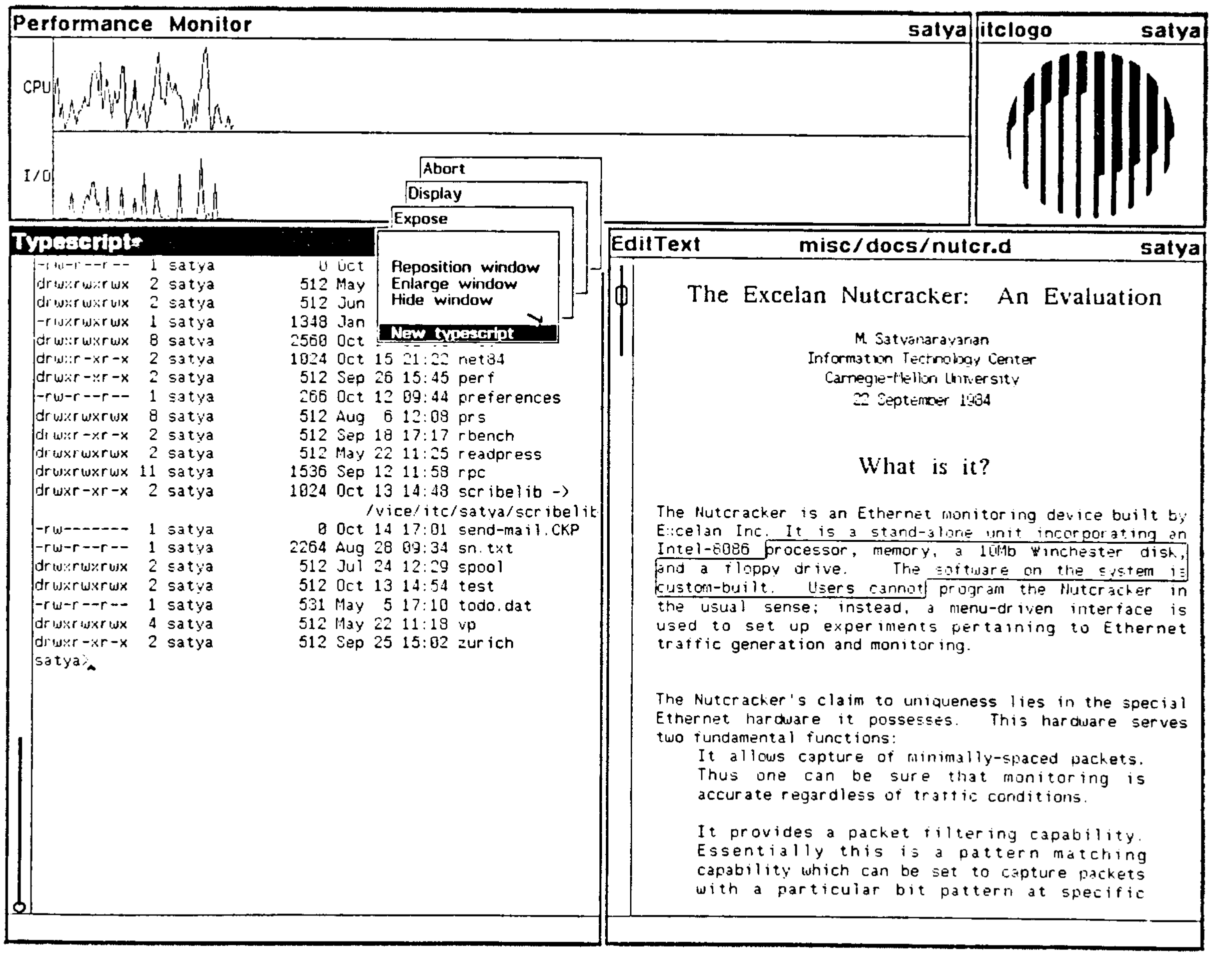


Figure 5-1: Window Manager Process Structure

Mouse movements are tracked by the window manager and converted into the appropriate cursor movements on the display. Keyboard typein is routed to the client in whose window the cursor was when the characters were typed in. Clients obtain information about mouse events and keyboard typein by querying the window manager. A pop-up menu system triggered by mouse clicks is provided. The contents of the menus, and the actions taken on their selection are client-specific. A subset of the menu items, such as reshaping a window or redrawing the screen, are acted upon by the window manager itself.

Clients have no control over the precise locations or shapes of their windows. These choices are either made heuristically by the window manager, or on demand by the user. The window manager does not possess adequate high-level knowledge about the contents of windows to be able to redraw them correctly. Hence, when a window needs to be redrawn, the appropriate client is informed via a software interrupt. It is the responsibility of that client 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 fact that clients cannot insist on specific window sizes or shapes, and may be called upon to redraw their windows at any time, enforces a programming discipline that makes clients relatively immune to specific display hardware characteristics. In effect, clients are being asked to deal with a new display whenever the size or shape of a window changes. The window manager itself will, of course, have to be modified to deal with new display hardware.

The window manager has been in use in the ITC for nearly a year, and has undergone a number of refinements. It is so versatile and useful that using it is the norm rather than the exception. Figure 5-2 is a snapshot of a workstation screen, showing the multi-process, multi-font, and graphic capabilities of the window manager.



```
-rw-r--r--  1 satya          0 Oct
drwxrwxrwx  2 satya        512 May
drwxrwxrwx  2 satya        512 Jun
-rwxrwxrwx  1 satya       1348 Jan
drwxrwxrwx  8 satya       2560 Oct
drwxr-xr-x  2 satya       1024 Oct 15 21:22 net84
drwxr-xr-x  2 satya        512 Sep 26 15:45 perf
-rw-r--r--  1 satya        266 Oct 12 09:44 preferences
drwxrwxrwx  8 satya        512 Aug  6 12:08 prs
drwxr-xr-x  2 satya        512 Sep 18 17:17 rbench
drwxrwxrwx  2 satya        512 May 22 11:25 readpress
drwxrwxrwx 11 satya       1536 Sep 12 11:58 rpc
drwxr-xr-x  2 satya       1024 Oct 13 14:48 scribelib ->
                               /vice/itc/satya/scribelib
-rw-------  1 satya          0 Oct 14 17:01 send-mail.CKP
-rw-r--r--  1 satya       2264 Aug 28 09:34 sn.txt
drwxrwxrwx  2 satya        512 Jul 24 12:29 spool
drwxrwxrwx  2 satya        512 Oct 13 14:54 test
-rw-r--r--  1 satya        531 May  5 17:10 todo.dat
drwxrwxrwx  4 satya        512 May 22 11:18 vp
drwxr-xr-x  2 satya        512 Sep 25 15:02 zurich
satya>
```

The Excelan Nutcracker: An Evaluation

M. Satyanarayanan
Information Technology Center
Carnegie-Mellon University
22 September 1984

What is it?

The Nutcracker is an Ethernet monitoring device built by Excelan Inc. It is a stand-alone unit incorporating an Intel-8086 processor, memory, a 10Mb Winchester disk, and a floppy drive. The software on the system is custom-built. Users cannot program the Nutcracker in the usual sense; instead, a menu-driven interface is used to set up experiments pertaining to Ethernet traffic generation and monitoring.

The Nutcracker's claim to uniqueness lies in the special Ethernet hardware it possesses. This hardware serves two fundamental functions:

It allows capture of minimally-spaced packets. Thus one can be sure that monitoring is accurate regardless of traffic conditions.

It provides a packet filtering capability. Essentially this is a pattern matching capability which can be set to capture packets with a particular bit pattern at specific

Figure 5-2: Snapshot of a Window Manager Screen

5.2. A User-Interface Toolkit

Built on top of the window manager, but distinct from it, is an ITC-developed library of graphical abstract data types called the ***Base Editor Toolkit.*** These data types are similar to classical abstract data types in programming languages [29, 13] in that they consist of encapsulations of data structures and operations on them. However, 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 which use the toolkit exclusively for their interactions with users are benefitted in a number of ways:

- The toolkit interface is a higher-level interface than the window manager interface. This relieves the application developer of many programming details.
- The user interfaces of programs that use the toolkit 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 honing the implementations of a small number of data types than by refining individual application programs.

The most basic data type in the toolkit 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 depends on the application program. A ***Scrollbar*** is a data type used in conjunction with a view on a document, and is used to make different parts of the document visible on the screen. The toolkit includes a family of data types referred to as ***Buttons***. These are labelled, rectangular screen objects each of which is instantiated 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 toolkit 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.

The toolkit is implmented as a library of subroutines in C. No language extensions have been made. It would be possible to incorporate the toolkit into C using a subclassing facility such as that described by Stroustrup [25]. This would make the programming interface more robust, on account of the stronger typechecking enforced by the subclassing mechanism.

Usage experience with the base editor toolkit has proved that it is indeed a valuable asset to the ITC. A number of application programs have been developed in a surprisingly short amount of time. Having learned the interfaces to a few of these programs, users are able to intuit the interfaces of the others quite well. Enriching the set of primitive data types, and making it easy for users to define their own data types are two areas where further work needs to be done.

5.3. Applications

Using the window manager and the base editor toolkit, a potpourri of applications which use the capabilities of a bit-mapped display have been developed at the ITC. The unifying theme of these applications is that they are of use to us in the course of our development work.

Examples of these applications are:

- a *bulletin board* browsing program, supporting an extensible set of interest groups.
- a *text editor* with dynamic formatting capability. This editor is superficially similar to a "What you see is what you get" editor, but differs from the latter in that it makes no attempt to produce a replica of a printed page. Reshaping an editor window automatically reformats the text to fit the new window.
- a special *preview* program that displays formatted text destined for a printer.
- an *access list editor* for manipulating file system access lists.
- a *personnel directory* containing the names and addresses of individuals.
- a *drawing program* which is built directly on top of the window manager and provides a reasonably sophisticated drawing facility.
- an *activity management tool* which assists individuals in keeping track of work items.

These applications have been in use for a few months and have come to form part of the repertoire of programs in regular use by the ITC.

An experimental icon-based interface to Unix, called ***Don Z***, is also being developed. Replacing keyboard typein almost entirely by mouse actions, Don Z provides users with a way to communicate with Unix in a less terse and cryptic manner than that required by Unix traditionally. Figure 5-3 shows a screen snapshot during a dialogue with Don Z. This system is operational but still undergoing development. With experience and refinement, it may well become the preferred mode of communication with Unix in the future.

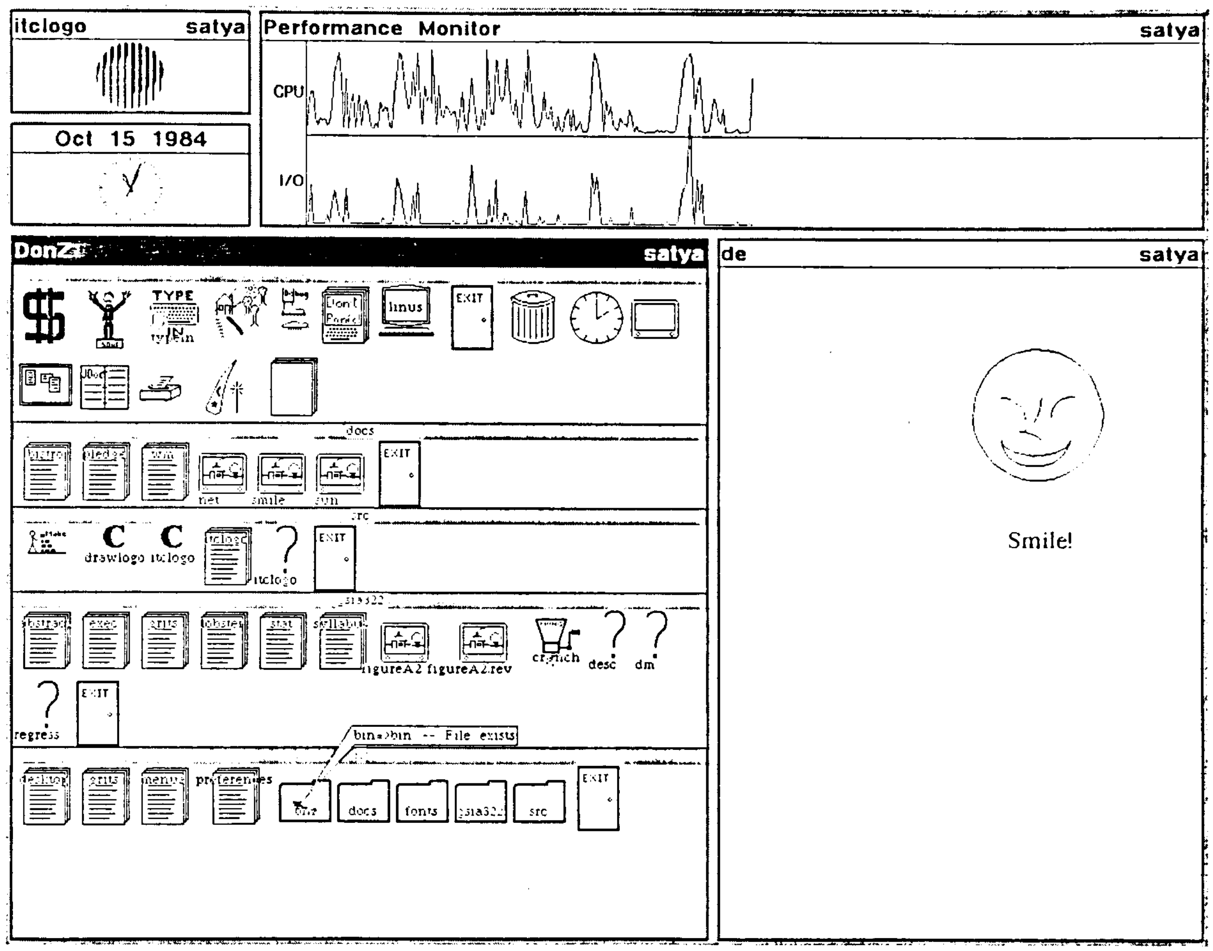


Figure 5-3: Don Z.: An Icon-Based Unix Interface

6. Project Status

The ITC project is currently at the midpoint of its expected development period. The major activities and accomplishments to date can be chronologically summarized as follows:

October 1982	IBM-CMU contract signed, establishing the ITC.
January 1983	Project started.
January to July 1983	Hiring, top-level structuring of project, definition of specific goals, architectural discussions.
August 1983	Development hardware and software obtained. These were Sun workstations running the 4.2BSD Unix operating system, with Ethernet networking support.

October 1983	File system architecture complete.
November 1983	First release of window manager in use.
November 1983	Prototype file system implementation design complete.
January 1984	First release of base editor toolkit available.
March 1984	First application program using base editor toolkit available.
July 1984	File system prototype available for use.
Present	30 users in the ITC use the window manager, base editor and file system on a daily basis. A variety of application packages have been developed on the base editor for applications such as bulletin boards, staff directories, and editing file system access lists.

Our current and projected future efforts include:

- *Pilot deployment of a prototype.*
  About 30 to 50 faculty members on campus are being selected for advance exposure to the prototype system. They will be supplied with hardware and software similar to that in use at the ITC. This pilot deployment will serve two functions: it will allow the faculty members to use the ITC system to develop educational software and, at the same time, give valuable user feedback about our design and implementation. This deployment is scheduled for November 1984.

- *Performance analysis of the file system.*
  Measurement of the current prototype is under way and is expected to yield information needed for a reimplementation.

- *File System Reimplementation.*
  The current implementation already gives us much of the functionality we require. It is noticeably slower than a stand-alone Unix file system, but is not so slow that it is unusable. For a prototype system, it is quite robust. The reimplementation will use the experience gained with the current prototype to produce a system that emphasises performance and reliability. We expect this implementation to be available in the spring of 1985.

- *Development of application packages*
  These packages will use the window manager and base editor for a variety of end-user applications.

- *Large-scale deployment*
  Faculty members are expected to receive workstations in the fall of 1985, and students are expected to receive them a year later. A whole host of activities, including wiring with LANs, porting of the ITC software to the final hardware, and designing appropriate administrative controls for the system are involved in this aspect of the project.

To summarize, therefore, the ITC currently has a small-scale, operational prototype of the system described in this paper. The next phase of the ITC's efforts is essentially directed at building a scaled-up version of the system, taking due account of our positive and negative usage experiences with the present system.

7. Conclusion

As mentioned at the beginning of this paper, the ITC is an institution created with the express purpose of designing and implementing a computing environment that will serve as a unifying presence in the educational, administrative, and social life of CMU. To meet this challenge, a system that represents the synthesis of personal computing and timesharing has been designed. The nature of the problem has necessitated the use of state-of-the-art techniques in local area network technology, distributed file system design, and user interface design. Using existing vendor-supplied hardware, a prototype has been implemented with a view to testing out the design. The experience to date indicates that the design is fundamentally sound, though refinements are necessary in a number of areas. As the system grows there will inevitably be many iterations over the design and implementation of various parts of the system. This is, of course, a characteristic of almost all large, heavily-used systems regardless of their specific goals.

In conclusion, it is appropriate to ask what is unique and noteworthy about this project. In many ways, the most fascinating aspect of the project is its scale. Never before has there been an attempt to integrate more than 5000 autonomous computational entities, each under the total control of an unconstrained individual. Scale is important not because bigger is necessarily better, but because it makes the building of a "real" system for "real" users that much harder. Reliability, security, and performance problems all conspire to make the design of such a system an intellectual challenge of first magnitude.

Acknowledgements

The work described in this paper represents the creative efforts of the entire staff of the ITC. The author is only one of the many individuals who have contributed to the ideas described here. Both CMU and IBM deserve credit for their willingness to chart a course into unknown waters, and for providing an excellent working environment within the ITC.

References

[1] Accetta, M., Robertson, G., Satyanarayanan, M. and Thompson, M.
The Design of a Network-Based Central File System.
Technical Report CMU-CS-80-134, Department of Computer Science, Carnegie-Mellon University, August 1980.

[2] Birell, A.D., Levin, R., Needham, R.M. and Schroeder, M.D.
Grapevine: An Exercise in Distributed Computing.
In *Proceedings of the Eighth Symposium on Operating Systems Principles.* Asilomar, CA, December, 1981.

[3] Boggs, D.R. and Taft, E.A.
FTP.
In Taft, E. (editor), *Alto User's Handbook.* Xerox Palo Alto Research Center, September 1979.

[4] Brotz, D. and Levin, R.
Laurel.
In Taft, E. (editor), *Alto User's Handbook.* Xerox Palo Alto Research Center, September 1979.

[5] Bux, W., Closs, F., Janson, P.A., Kummerle, K., Muller, H.R. and Rothauser, E.H.
A local-area communication network based on a reliable token ring system.
In *Proceedings of the International Symposium on Local Computer Networks.* Florence, Italy, April 1982.

[6] Crane, R.C. and Taft, E.A.
Practical Considerations in Ethernet Local Network Design.
Technical Report, Xerox Palo Alto Research Center, October 1979, revised February 1980.

[7] Defense Advanced Research Projects Agency, Information Processing Techniques Office.
RFC 791: DARPA Internet Program Protocol Specification
September 1981.

[8] Day, J.D. and Zimmermann, H.
The OSI Reference Model.
Proceedings of the IEEE 71(12), December, 1983.

[9] Howard, J.H. (Editor).
ITC File System Design.
Technical Report, Information Technology Center, Carnegie-Mellon University, September 1983.

[10] IEEE Project 802 Committee.
Local Area Network Standards (Draft Standard 802.5).

[11] Jones, A.K. and Gehringer E.F. (Editors).
The Cm Multiprocessor Project: A Research Review.*
Technical Report CMU-CS-80-131, Department of Computer Science, Carnegie-Mellon University, July 1980.

[12] Lazowska, E.D., Zahorjan, J., Cheriton, D.R. and Zwaenepoel, W.
File Access Performance of Diskless Workstations.
Technical Report 84-06-01, Department of Computer Science, University of Washington, June 1984.

[13] Liskov, B., Snyder, A., Atkinson, R. and Schaffert, C.
Abstraction Mechanisms in CLU.
Communications of the ACM 20(8), August, 1977.

[14] Metcalfe, R.M. and Boggs, D.R.
Ethernet: Distributed Packet Switching for Local Computer Networks.
Technical Report CSL-75-7, Xerox Palo Alto Research Center, May 1975, reprinted February 1980.

[15] Nelson, B.J.
Remote Procedure Call.
PhD thesis, Department of Computer Science, Carnegie-Mellon University, May, 1981.

[16] The Task Force for the Future of Computing, Alan Newell (Chairman).
The Future of Computing at Carnegie-Mellon University.
February 1982.