CMU-ITC-83-031

ITC File System Goals

12 September 1983

File System Group

Information Technology Center
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

CONTENTS

INTRODUCTION

This is the file system part of an overview of the ITC's project goals. Its purpose is to tell you what using the VICE file system will be like, so that you have a chance to influence our design now and we have a target to shoot at.

VICE provides communications and shared file storage for the ITC network of workstations. Communications, including the local area network and also the set of "gateways" to existing campus computers, is discussed elsewhere, as is the VIRTUE workstation. Here we describe the file system, including storage and movement of files, protection and authentication, accounting and space control, dealing with network failures and independent operation, and certain applications such as mail and bulletin boards.

Ordinarily you can write programs and manipulate files as if the file system were stored on a single very large computer system. The workstation will contain interfacing software whose job it is to insulate you from the details of network interfacing, copying files to and from the workstation, and so on. However, the network won't be completely invisible:

- You'll notice performance differences because locally stored files are easier to get to.
- You can, if you wish, control file placement, bypass the interfacing code, or even build your own workstation.
- The file system will attempt to provide (possibly limited) services even if the workstation is disconnected from the network or the network itself is partitioned.

Therefore, a little terminology about the structure of the network is in order.

The network will be organized into interconnected "clusters" of workstations. Each cluster connects some number of workstations and at least one "cluster server" machine, which stores files and runs other VICE applications. For security reasons, cluster servers run only VICE applications. The hardware and software of the attached workstations may differ, but they can all use the services provided by VICE if they support the underlying communication and application protocols.

Within this framework, there are two file system interfaces: the VIRTUE interface, through which programs and user commands running in the workstation see files stored in VICE and/or locally, and the VICE interface, used by the workstation (and potentially by any machine connected to the network) to store, share, and retrieve files. You will seldom deal directly with the VICE interface, which is really intended for machines, and will usually deal with the VIRTUE interface by running programs which read and write files.

What will using the file system be like? This section attempts to give you simple examples, with most of the detail suppressed, of what things you can do with the ITC file system.

ORDINARY FILE ACCESS

When you start using a workstation, you "log on" by providing a password and your name. The workstation goes through an authentication procedure to establish a secure connection with VICE. Since only your workstation can use the connection, and you're in control of the workstation, VICE treats the connection as your ticket to your files. (It will be up to you to "log off" when you're done, to prevent somebody else from misappropriating the connection and masquerading as you.)

When you first use a file, the workstation will copy it automatically to its local "cache" of files. Thereafter it will use the cached copy directly, checking with VICE to verify that nobody else has updated the master copy. If you modify the file, the workstation will copy it back to VICE automatically. If you stop using the file, the workstation will eventually discard the local copy in order to make space for other files.

Some widely-used files, such as compilers or popular applications programs, will be replicated at every cluster server. VICE will update such files periodically, for example once a day. You can force an update if you really must have the most recent version.

SHARING

You can use your files from any workstation, regardless of where they're actually stored. For example, you can log on to a public workstation anywhere on campus and use your files normally, although possibly with a performance penalty if the files must be moved a long distance.

Only you have access to your files to start out with. You can, however, allow other users or groups of users to perform selected operations on your files. For example, a research project can set up a shared set of files which any member of the project can read and update.

MAIL AND BULLETIN BOARDS

Mail and bulletin boards are a widely-used example of file sharing. VICE will support them through its tree-structured directory. Every user who wishes to receive mail will have a "mailbox" subdirectory into which other users are permitted to store messages in the form of files. Only you can remove and read messages from your mailbox.

Bulletin boards will resemble mail, except that a whole group of users can read the messages. Stale messages will be removed by the bulletin board's owner.

VIRTUE will provide application programs to generate, categorize, and read mail and bulletin boards; the file system simply stores them.

LOCAL FILES

You can force files to be stored only in your workstation by placing them in a special "/local" subdirectory of your workstation's file system. This provides some additional security for very sensitive files, but you won't be able to get at them from any other workstation. A more important application is to store the programs and data needed to get the workstation started and connected to the network when its power is turned on.

INDEPENDENT OPERATION

Assuming your workstation has sufficient local storage to hold your files and programs, you should be able to use it even if it is disconnected from VICE. This could happen if you take the workstation off-campus, or if there is some failure in the network or cluster server. This is a special case of what is called "partitioned" operation, meaning that the network has been divided into two or more independently operating parts which can't talk to each other.

Your workstation will retain file updates until reconnected to VICE, then merge your local changes with any that happened in the rest of the network. We expect that conflicting updates will be relatively rare, but will attempt to detect them by keeping some sort of update log.

The ability to operate independently requires that your workstation have a local disk with reasonable performance. We may also provide for diskless workstations (which would not be able to run independently) in order to reduce the minimum cost of a workstation. This decision depends on technological and performance considerations not yet available. Diskless workstations would depend on the cluster server to store almost all of their files.

The following sections deal in somewhat more detail with the various major components of the file system.

VIRTUE INTERFACE

The VIRTUE file system looks and behaves like a single-site Unix file system, with a tree-structured directory. The VIRTUE user interface is expected to provide a convenient and easily understood set of user-level commands to manipulate files. The file system interface proper is in the form of "system calls" through which application programs can create, locate, read, write, close, and control files.

With one exception, the VIRTUE file name space is identical to that of VICE. The exception is the subtree "/local," which contains files strictly local to the workstation. Symbolic links from the VICE name space to files in "/local" may be used to achieve workstation-sensitive file name interpretation.

The VIRTUE system calls are intended to resemble standard Unix system calls closely enough that it is easy to port Unix application programs to the workstation. Exactly how compatible the interface is depends on which version of Unix you're talking about; there's no universal standard. Some incompatibilities known at the present time are:

1. Symbolic links only.

   Most Unix systems provide "physical links" between files. The key property of physical links is that if somebody else replaces or deletes a file to which you have a physical link, you retain access to the *old* version of the file. By contrast, a symbolic link is no more than an alternate name for a file. Changes to the file change it under both names. We feel that physical links between cluster servers will be too hard to implement, so we will provide only symbolic links.

2. Protection.

   While maintaining reasonable compatibility with Unix, we may decide to implement a more flexible and/or robust protection system than that of Unix.

3. Directory Structure.

   Programs which read directories as files, expecting a certain format, may fail or run slowly.

4. Concurrent Updating.

   It may not be possible for two programs to update or append to the same file concurrently.

VICE INTERFACE

The VICE interface is designed for simple and efficient communication with VIRTUE and possibly other programs or workstations. Thus it's of interest only if you intend to bypass the VIRTUE interface and write a program which talks directly to the network.

The VICE file system stores files and identifies them with a Unix-like hierarchy of directories. Naming conventions and file formats follow Unix; for example file names are case sensitive, and a file is treated as an arbitrary sequence of 8-bit bytes with no pre-specified internal structure.

Our strategy is to transfer whole files between the workstation and VICE, so the interface is based on file Fetch and Store operations employing an as-yet unspecified file transfer protocol. In addition to these basic operations, a workstation will be able to lock and unlock files, to read and write file descriptor information, and to create and destroy empty subdirectories, files, and symbolic links between files.

At a later time we may extend the VICE interface to include not only whole file transfer but also a way to open a remote file and then read or write pieces of it. Such a facility may be required for efficient access to large files, such as data bases. It is also one of way to implement diskless workstations.

PROTECTION

NOTE

This section has not yet been agreed to.

Since we feel that the standard Unix protection mechanism is inadequate for a large, diverse user community, we propose to replace it with an access-list system.

In such a system you associate with the objects you wish to protect a list of other users (or groups of users) and the operations each is allowed to perform. In the VICE file system, the protected objects are directories, and the operations are:

create store new files in the directory

machine. Servers which the user may use will have to be able to find the correct Authorization Server to ask for validation, perhaps by means of a cluster-address part of the user's network address.

1.3.1.1 Authentication Data Bases

The servers store permanent information about each user in a User Authorization File (UAF), which is a single file stored by the File Server in a "system" directory (such as "/etc"), read and modified only by Authentication Server facilities (although, for maintainability, it should be a text file, suitable for modification with an editor). There should be only one such file in the network (since users can not be restricted to using workstations on a particular cluster) although while running there will presumably be instantiations of the UAF on many or all of the cluster machine's file servers. The UAF contains one record for each computer network user, containing:

username The computer-oriented user identification, as assigned by the central administration (the Registrar for students, Personnel for faculty and staff). It should be short enough to be conveniently specified by users but long enough to be unique over the lifetime of the usernames. Whether it is "HRU17" or "HughRalphUser" it should be the first thing in the record, since it is the search key.

password The password for the user, in some encrypted form.

accounts A list of computer usage account codes, to which the user is authorized to charge his usage. The format of these account codes is yet to be discussed; at present they are four-character codes such as "R602."

true name The user's real name, as he is known to the university. This will be needed for personal-name to computer username lookups, for such things as mail delivery; it will also be useful for headings on computer printer output. Like everything else in this record, the user should not be able to change this without someone responsible making sure it is really his name.

There will also have to be a relatively small Account Classification File, which tells the server the "account type" for each account code; different accounts will need to have different billing rate structures for some servers, so this administratively-assigned information will need to exist in another common file.

account The account code.

type The account type; at present they are single character codes.

The servers also store dynamic information about each active user in a User Identification Block (UIB), maintained in the virtual memory of each Authorization Server:

username As in the UAF.

key The access key assigned by Logon.

account The account code for this session.

type The account type code.

Other things which might be useful in the UIB are:

net address The network address of the workstation the user is using, if such information would not simply be in the name server.

directory The path of the user's top-level directory in the file system, if such things might not be based simply on the username.

1.3.1.2 Logon function

The user ordinarily meets the Authentication Server only while establishing a connection to the network, for example after turning on the workstation or when passing a shared workstation from one user to another. Once such a connection is established, the servers, network mechanisms, and workstation software cooperate in maintaining secure and authorized access without bothering the user.

The message which the user workstation software sends to the Logon function is his computer username, his password, and the account code he wishes to use for this session. The server locates the entry in the UAF, validates the password provided against the password in the UAF, and validates the account code requested against the list of allowed account codes. Having validated the request, the server builds a UIB, fills it in with the user's username, account code, account type (obtained from a separate file), and a unique number to serve as a validation key (the validation key should contain a cluster number so that servers can quickly get back to the correct Authorization Server). The server returns the key to the workstation, which the workstation will use as its identification for all future service requests. This should allow the workstation software and the servers to cooperate in maintaining secure and authorized access, without the user having to provide servers with his real password, and without his workstation having to ask him for his password again. The Logon function also calls the Logon function of the Accounting Server to initialize its dynamic data base.

1.3.1.3 Validate function

This function is called by servers to validate a request made to them by a user workstation. It takes a username and a key and returns the contents of the UIB if the key matches. If there were nothing in the UIB of value to servers, it could merely return ACK or NAK, but servers will want to know such things as the account type to determine their billing rates.

1.3.1.4 Logoff function

Since maintaining an active user within the memory of the server machines is a use of scarce resources, the user (or the workstation software) is responsible for informing the Authentication Server that the user will be making no more service requests. Future service requests for that key will be rejected. Also, there will be dynamic information stored in the UIB and other context blocks in other servers which should be written back to disk data bases; nothing presently in the UIB requires update of the UAF, but the Logoff function of the Accounting Server should be called. Unusual situations may require some sort of forced logoff, for example when a server is being shut down for maintenance. Thus a key may be revoked, or service may be refused, after suitable warnings.

1.3.1.5 Administrative functions

Whatever specific data is maintained by the Authentication server must be updatable both by administrators and (within limits) by the users themselves. Thus there will be functions to add, update, remove, and review current registrations. These will only operate on the central machine, where the relevant files will reside.

1.3.2 ACCOUNTING SERVICE

Network services require finite resources which must be accounted for in order to allow equitable scheduling and charging. Examples of such resources are file system space, communications bandwidth, and server machine time. Resources local to the workstations need not be accounted for, although there is a gray area in charging for use of proprietary programs.

The charging rate for a service should be proportional not only to the amount of resource consumed but also to the contention for it, so that rich users can't monopolize the network. (Actually, this should be a rule for the individual servers, since they set their own rates.) To avoid undue charges at busy times, users should be able to control the maximum rate at which they spend money. (Note that this may be ineffective where there are queued services, such as print requests, or regular charges, such as disk storage.) Servers should reserve money prior to granting service.

It would be simplest to simply maintain in the UAB or UAF a one-word count of the amount of money remaining in a user's account, and do nothing more than decrement it as charges arrive at the server. However, customers require itemized billing to provide justification for the amounts they must pay, and to give them useful information on the resources which they are using. Given only a lump sum charged to "computer network usage," the user would have no way to decide whether to use off-peak hours, or cheaper file storage, or make intelligent budgeting decisions. Further, funded research contract accounts have historically needed itemized billing information to justify their charges to their sponsors.

There should be one copy of the Accounting Server running on each of the cluster machines. Each will write accounting transactions to its own files, which will presumably reside on the file server of the same cluster machine; thus, each cluster should be able to function without dependence on the rest of the network being operational. A maintenance function on the central machine will regularly bring in accounting transactions from the cluster machines.

An issue: Should users be allowed to be logged in more than once? Within the same cluster (and thus known by the same Accounting Server)? In different clusters (thus requiring some cooperation from a central server if it is essential that users not exceed their allocations)? In either case, there will have to be record-level locking facilities in the file system, so that a server on one cluster machine can modify a record in the accounting data base without another Accounting Server interfering.

1.3.2.1 Accounting Data Bases

There will be an Accounting Data Base (ADB) file, stored by the File Server in a "system" directory, read and modified only by the Accounting Servers. It seems as if there would have to be only one copy over the entire network, or else accounting servers on different clusters would have conflicting ideas regarding how much money a given user had remaining, and yet each accounting server must be able to function when cut off from the rest of the network. Perhaps it should normally use the central file, but if the file cannot be accessed the server should make

some reasonable assumption about how much money the user has left, and queue up within itself the modifications it wants to make to the user's record when the necessary central file becomes available.

The ADB contains one record for each username/account code pair:

username Username.

account Account code.

spent Funds used by the user for that account.

billed Funds used up through last billing period. (Updated only by Maintenance function.)

allocated Funds allocated by the account administrator. The user can not spend more than this amount of money.

It might be nice to keep all of this information in the UAF:

The total number of system data base files would be reduced.

The functions of the Authentication Server would be able to maintain the information.

However, by keeping a separate data base

The formats of the records can be known to fewer programs and fewer programmers, and thus can be more reliably developed and maintained.

The UAF, as described above, contains only permanent information; it should not be necessary to make changes to a user's UAF record (as now defined) more than a few times a year. The ADB, on the other hand, will change very often, certainly at the end of every system usage session. Keeping a separate ADB reduces the hazards of often-changing files.

There will also be an Accounting Transaction File (ATF) containing a record of each billable transaction charged for the session (actually, only for each class of identical transactions for the session, since there will be very many transactions from very few services in the ordinary case). This will likewise be a file in the File Server, read and appended to by the Accounting Servers, but in this case each cluster machine should have an independent transaction file.

username Username.

account Account code.

type Account type.

commodity Unique commodity code for the service provided, assigned by the central administration.

charge basis A service-specific indication of the basis for the billing rate used. Some servers will charge at different rates for different users, or for different times of the day. For each commodity with such variable rates, there will be a list of charge basis codes to allow the end-user to see the justification for the charges made.

extra An extra word of information, specific to the particular commodity, such as "pages printed" for printer services.

charge The number of cents charged for the service.

There will also be a dynamic User Account Block (UAB) built in the virtual memory of each Accounting Server for each active user session.

username Username.

account Account code.

type Account type.

spent Funds used in the current session (set to zero at Logon).

allocated Funds remaining in the allocation (set to ADB.allocated minus ADB.spent at Logon).

rate The maximum rate at which the user wants to spend money. This would be something like "cents per minute."

check time The last time the spending rate was computed.

check rate Some time-weighted measurement of the rate at which the user is spending money.

reservations A pointer to a linked list of Accounting Reservation Blocks (ARB):

- **flags** A flag indicating whether this is a known-server transaction or a user-to-user transaction.
- **server** Centrally assigned unique identification of the server making the reservation.
- **commodity** Centrally assigned unique identification of the service bring provided; if user-to-user transaction, buyer-specified code.

serial Server-specified serial identification number for the reservation.

seller code Buyer-provided identification of the user selling the service in a user-to-user transaction.

price The amount of money reserved.

time limit If provided, a time limit, after which the service will be said not to have been provided.

charges A pointer to a linked list of Accounting Transaction Blocks (ATB), one for each class of identical service requests; e.g., there will be one ATB for mail messages, one for file server data transfers, perhaps another for file server file deletions.

server. The server's unique identification

commodity The commodity code.

charge basis The basis for the billing rate.

count A count of the number of instances of this service.

extra A word of additional data, such as pages printed; each service transaction will accumulate into this word.

subtotal The amount of money spent on this service.

1.3.2.2 Logon function

This is called by the Authentication Server on the same cluster machine to access the correct entry in the ADB and to build a UAB. (If it is desired that users get warnings if their accounts have reached various warning levels, this is the place to do it.)

1.3.2.3 Reserve function

A system server (i.e., one known to the network administration) uses this function to verify that a user has an adequate balance for an

anticipated service request, and to reserve that amount of money for the server, to be unusable by the user until the service is completed or cancelled. The server uses the cluster-number portion of the key which the user provided it to contact the Accounting Server on which the user is logged on. The server specifies a network-unique identification for itself (not a network port number, but an administratively assigned number meaning, say, "Cluster 3 Mail Server"), the commodity number for the service to be provided, a server-unique sequence number for this particular transaction, the expected (maximum) price for the service, a server-defined charge basis, the username, and the key.

The Accounting Server verifies that the server is authorized to make charges for the provided commodity code, then checks whether the user has funds available, and then whether the user has been using money faster than he wishes to; if any of these are not met, an error is returned to the server. The Reserve function then builds a new ARB, fills it in with the supplied data, links it into the UAB.reservations list, adds the expected price to UAB.spent, and returns an acknowledgement to the server.

Until the server completes the function and indicates the final, exact price, the money must be assumed to be unavailable to the user. In the case of server failures or timeout, the reserved quantity is returned to the user's account. (The server may have to provide some timeout period).

1.3.2.4 Debit function

After completing a service, a server performs the actual debit using the Debit function, supplying the username, the key, the transaction sequence number, the commodity number, the auxiliary data if desired, and the actual price. Debit locates the proper ARB, subtracts the actual price from ARB.price, subtracts that refunded amount from UAB.spent, locates an ATB for the server and commodity (or creates it if a suitable one does not exist), adds the transaction price and auxiliary data, and removes the ARB.

It may be that the Accounting Server has crashed since the Reserve was done. In that case, there will be no ARB, and there may be no UAB. If there is no ARB, Debit should simply perform the billing function. If there is no UAB, perhaps it should return an error indication to force the user to log on again?

1.3.2.5 Purchase function

Users can transfer money to other users in an organized manner. First the buyer confers with the seller and agrees on a price, and perhaps on a unique code by which the buyer can identify the seller to the Accounting Server; this may take place within the computer system or outside of it. The buyer calls Purchase on his own cluster machine's Accounting Server to authorize the transaction, supplying the agreed code number for the seller, an optional time limit, an expected price, a buyer-specified "commodity number" to identify the transaction for himself (although this is an unofficial service, the buyer should be able to keep track of his usage), and a transaction serial number. The Purchase function ensures that the funds are available, builds an ARB, fills it in (specifying "user-to-user" as the server identification), adds the price to AUB.spent, and returns the transaction serial number to the buyer.

1.3.2.6 Sell function

A user process selling an unofficial service uses this function to actually transfer the funds to its own account. After obtaining the seller code from the buyer, and the transaction serial number, the seller performs the requested service, and then calls Sell on the buyers's cluster machine, providing the seller code, the buyer's username, a charge basis or extra data if desired, and a final price. Sell finds the UAB corresponding to the username and seller code, then locates the ARB according to the seller code. It then builds or updates an ATB based on the information, just the same as Debit, updates UAB.spent, deallocates the ARB, locates the UAB for the seller, and decrements the amount spent from the seller's UAB.spent. (There may well have to be a more complex mechanism; after all, sellers may want to get real money instead of computer usage out of their products.)

1.3.2.7 Charge function

There must be a way to bill a user for a queued service, such as printing, which will often be billed after the user has left the system. In this case the server must provide the userid, account code, charge basis, extra data, server code, commodity code, and price; Charge will see if the user is logged in (anywhere?), and if not it will do the entire billing function (writing of transaction file record and update of ADB.

1.3.2.8 Logout function

This is called by the Authentication Server to write ATB's to the ATF, to update the ADB, and to deallocate the UAB.

1.3.2.9 Maintenance functions

There must be maintenance functions to modify the ADB and to copy or send the ATF for processing.

1.3.3 FILE SERVER

The primary concern of the file server is availability and dependability, like the transport services. Once a file is delivered to the file server it will not be lost. The file server will ensure that no two users are updating a file at the same time. It will optionally be able to prevent a file being updated while it is being read. It prevents deadlocks by never allowing a user to wait for a file. The functions of the file server are:

1.3.3.1 Put/Get

A file copy operation between a server and a user work station. The unit of transfer will be files. There must be room on the user work station for the file.

1.3.3.2 Transfer

A file copy operation between two user's work station. The unit of transfer will be files. There must be room on the target user work station for the file. This service will be used by the print and mail servers as well as others.

1.3.3.3 Partial File Operations

This is a follow on service that will be needed for use by the AFWS. It will provide the ability to move only part of the file to a user machine. This could be used to bring the data up a page at a time or in larger or smaller pieces depending on what the user computer is doing. The functions provided will be described in the initial document to ensure that they will satisfy the needs of the AFWS.

1.3.3.4 Directory

A set of services to find out the names of files in the system and some of their attributes. The level of function is at least equivalent to the IBM PC DIR function. This service will also check to see that any new name is unique within the system. It would only tell a user about files that he was allowed to access.

1.3.3.5 Access Control

A means of limiting access to files to specified users. It must allow at least a distinction between read and write access to the files. This service works with the authentication services.

1.3.3.6 Resource Control

A means of controlling the amount of space used by each user. This would prevent a user from flooding a server system with data requests.

1.3.3.7 Archive Control

A means of moving data to lower cost storage media. This service would also choose where data was to reside. It would move data from a server to the central system and from high speed storage like DASD to lower speed storage like laser disks or tape.

1.3.3.8 Metering & Measuring

Ability to measure data flow and control data flow to specific users. This would be used to keep control of individual user machines as well as recognizing when loops needed additional servers or no longer needed all the servers that they had. It would be useful for keeping track of statistics on the operation of the network and the placement of files.

1.3.4 MAIL SERVER

The mail server will provide both brief messages and electronic letter services for user to user communication. The mail server will use the accounting server to bill for its services. The charge will depend on the size of the message and the number of users that are to receive it.

Copies of the mail server will run on each cluster machine. All servers must be able to access files anywhere in the file system.

1.3.4.1 Message function

The message function sends a brief (one-line) message to a user. It is expected to be used both by other users ("Ready for lunch, Jim?") and by network servers ("You have incoming mail."). If the recipient is logged on, a connection is established to a server (whether written by the ITC or by the user) running in the user's workstation and the message is passed to the server. If the recipient is not logged on or his server is not receiving, the message is discarded and a negative reply ("West not logged on.") message is returned to the sender.

Assumption: Each active user can be found by a network address, and network connections may be made from a system server to a process in the workstation at the server's request.

1.3.4.2 Message-receive function

This process, running in the user's workstation, will do something useful with brief messages sent through Message.

1.3.4.3 Mail function

The mail function transmits a formatted message from one user to another, the local recipient(s) specified by computer username or by unique personal name; the server adds the username of the sender to the "From:" line and places the message in a file queue of messages to be delivered. The mail function bills the user immediately for the message, based on its length and the number of recipients. The recipient need not be logged on; if he is, a server on his workstation (ITC-written or user-written) will be called to say that there is new mail available. If the receiver is on another network, it is routed to the appropriate gateway.

There is the problem of people on different clusters. Probably the mail should migrate rather quickly to a mail server on the central machine. If the recipient is logged in, perhaps it should be routed to the mail server for his cluster; but then, he might not read it immediately, so going to the central machine would be better. If someone on cluster 1 sends to someone who at the moment happens to be logged in at cluster 3, the recipient (or his mail program or mail server) would have to know to ask cluster 1's Receive function for mail.

Additional capabilities which might be good to have: Mailing lists, automatic forwarding, bulletin board support.

1.3.4.4 Receive function

The Receive function, when called from a user workstation, searches the queue for a message to that user and returns it to him, deleting it from the queue.

1.3.4.5 Workstation programs

Although not part of the network services proper, the workstation software will play a key role in handling mail. It will send mail, and will receive mail from the mail server and store it in the user's mailbox. It should use standard facilities to look at and to edit the mailbox. It should be able to log all outgoing messages. It should acknowledge received messages upon request.

1.3.5 PRINT SERVER

The print server prints files, by queueing them to the many printer servers. From a network services standpoint, we are concerned only with delivery of files to the printer servers and not with formatting, which should be dealt with under the heading of user interfaces.

This is a queueing system; the files are not physically copied to the printer server until the data are needed.

There will be print servers on each cluster machine, which receive requests from users and route them to an appropriate printer server. There will also be printer servers on the machines which physically have the printer(s), which maintain their own queues and control their own physical printers; the printers will have different capabilities (e.g., graphics or special fonts). The printers will be located at different points in the network (e.g., some may be on various loops, others at the central site). The ability to specify which printer to use for either physical location or printer capability will be included in the print function. When a print job is complete the user will receive notification if requested, through the Mail Server's Message function.

The printer servers will use the Accounting Server to charge for printing. The parameters used for charging will depend on the capabilities of the printer used, the particular printer facilities requested (e.g., paper type, special operator handling), the size of the request (number of pages printed), and the contention for the the printer. The print server will have the ability to do load balancing between printers, where the requests allow; printer servers will be able to send requests to other acceptable printer servers.

There will have to be a system-wide Printer Definition File, so that all the print servers know which printer servers have what characteristics.

1.3.5.1 Printer server data bases

To manage printing requests, there must be a queue in each running print server and printer server, with one entry for each request.

sequence A unique number identifying the request.

username Username of the user requesting the print.

account Account code for the request.

classification Whatever classification requirements the user might desire.

job options Parameters for the entire print job.

- **notify** A flag, set if the user wants notification when the job completes.

files A list of files to print and options for them.

- **file name** The path to the file to print.
- **options** Parameters for that file.
  - **delete** A flag, set to delete the file after successful printing.

1.3.5.2 Print function

This causes a print to be placed in the queue of an appropriate printer server, according to the user's parameters, and considering other scheduling requirements they might have. It will send back to the user a print sequence number.

TBS: Contents of print server-to-printer server message.

There must also be a queue within each print server, since it may not be immediately possible to send the request through the network to the desired printer server (it may be down or the network may not be working).

1.3.5.3 Cancel function

Given a print sequence number, this will find the print server currently holding the queued print and tell it to cancel it.

1.3.5.4 Query function

Given some search criteria, this will find all matching print requests and return information on them.

1.3.5.5 Operator function

There will need to be services to allow an operator to control the print server and the printer servers.

1.3.5.6 Printer Server functions

The printer servers, though not accessed directly by user processes, must have several elements: Queue function, Cancel function, Query function, Operator function, Printer function.

1.4 IMPLEMENTATION & DESIGN

This part of our project, together with transport services, is the part that must bear the brunt of thousands of users. Therefore, we are being very conservative about what is included. The general rule is to make the facilities on the work station do anything that can be done for an individual user, and put only those things that involve multiple users in the common system; i.e. communication and shared files. In particular, it is extremely important that this service be dependable. If a tradeoff must be made between functionality and dependability, it will be in favor of dependability.

When I refer to common system services I am refering to all of the services that are provided by the various cluster servers and central systems.

There will be three types of machines in the network, and each will have different functions. The machine type will be determined by function, not by the model number of the machine. The machine types are:

1. User work stations

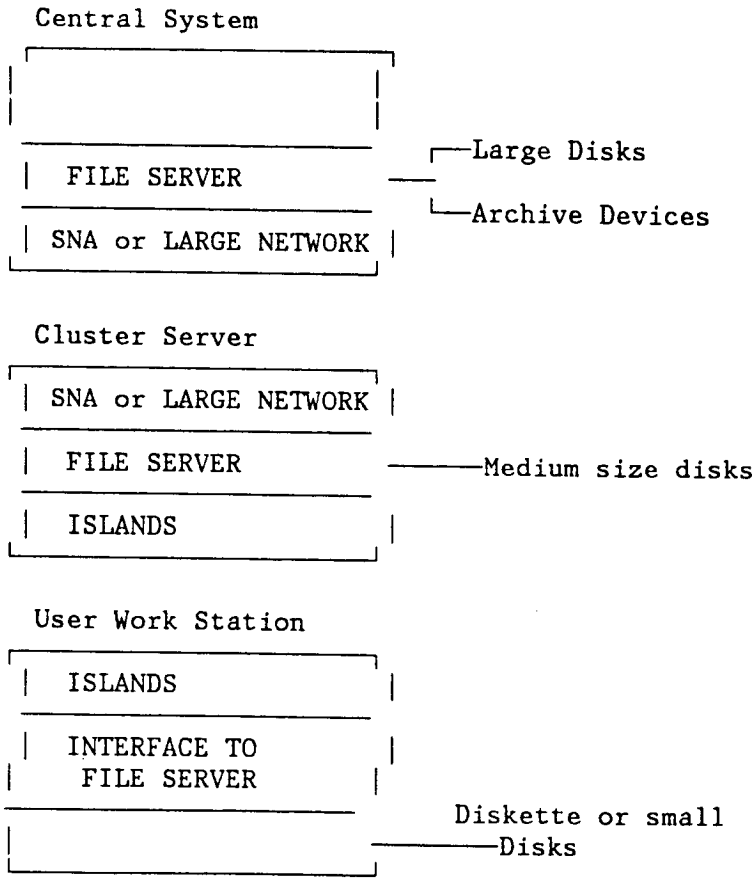
   User work stations are the personal computers used by the users. There will be many of them.

2. Cluster servers

   Cluster servers are the machines that exist to provide specialized services. There will be fewer of them. File servers are an example of server machines. There could be many types of server machines added over time.

3. Central system

Central systems are large machines that exist at the core of the network. They are the anchors for the network and will contain such things as archive storage and central processing for name resolution.



Relation of different machines
Figure 1

The early work, will be to develop and test the file server code. This is where the in use data is stored, and where the majority of the unknows exist. By focusing on the problems at this level, we will face our most difficult file server problems early.

It should be noticed that any disks or diskettes on the user work stations are not considered part of the file system. This will probably be a controversial decision, so it is being pointed out here.

1.5 MILESTONES:

- 3/83 to 4/83 General outline of common system services
- 4/83 to 6/83 Interface specification complete, prototype interface implemented on a stand alone work station
- 6/83 to 9/83 Single server machine up, providing service to multiple work stations.
- 9/83 to 12/83 Multiple servers demonstrated
- 3/84 All ITC using common network services
- 6/84 Additional user community served, total of 50 users
- 9/84 Additional users acquired, total of 200 users

1.6 DEPENDENCIES:

- The transport interface must provide a generic way to reach the local file server.
- 4/83 work station available for programming (e.g. PC with SRITEK board)
- 6/83 Machine available for cluster server (e.g. SUN or PERQ with large disk)
- 6/83 LU_6.2 transport interface available

ITC Transport Network White Paper

June 22nd, 1983

John Drake
Don Smith
Bryan Striemer

Information Technology Center
Carnegie-Mellon University
Schenley Park
Pittsburgh, PA 15213

GIBM & ITC CONFIDENTIAL --- PRELIMINARY DRAFT

IBM CONFIDENTIAL

ITC DEVELOPMENT

CONTENTS

1.0 INTRODUCTION

This white paper sketches the logical design and current implementation strategy for communications in the system developed by the ITC for Carnegie- Mellon University.

1.1 DOCUMENT STRUCTURE

This white paper is organized in four major sections:

1. Introduction

2. Requirements and Objectives: This section will describe the scope of communications in the ITC system and sketch the requirements for availability, performance, and interconnections.

3. Logical Design: This section will describe the logical structure, functions, and programming interfaces used in the communications network.

4. Implementation Strategy: This section will outline the means by which the logical design can be achieved, including dates at which significant components will be available, plans for building these components, and dependencies on other elements of the ITC development plan as well as external dependencies.

2.0 REQUIREMENTS AND OBJECTIVES

Our goal is to produce a data communications network to support as many as 10,000 powerful personal computers. This entails the operational support of a very large network providing services to an even larger community of users.

It is clear that this network will be unique in scope -- both in terms of size and variety of leading-edge uses. Thus the primary emphasis of the network should be high availability and performance. The next most important requirement is for meaningful interconnection and communication among a large number of software and hardware elements, including ones not developed by IBM or the ITC.

2.1 AVAILABILITY

Essentially the network must be as available as a public electricity utility. It must be possible for a user to "plug" his work station into the network at any time and have access to enough services to perform a guaranteed minimum level of useful work. Since individual "plugs and wires" may fail, the user may have to move his work station to a different plug or move himself to a publicly available work station.

2.2 PERFORMANCE

Performance, as observed by the work station user, must be such that a guaranteed amount of useful work can be accomplished in a given time. Response times experienced by the user should be fast enough to augment human acitvities.

Because performance is a critical requirement of the system and involves all elements (not just the communications network), it is recommended that a system performance task group be established in the ITC. This group would be responsible for configuration design, considering trade-offs among bandwidth of the local area network, service machine MIPS, and storage device capacity and access rates.

2.3 INTERCONNECTIONS

All ITC work stations and service machines should support a defined communication protocol allowing full interconnection among themselves.

Interconnection with IBM products that provide functions needed in the ITC system (e.g., application software, printers, image devices) should be provided. Interconnection with existing networks at Carnegie-Mellon University is required for migration of users to the new system.

3.0 LOGICAL DESIGN:

3.1 STRATEGY:

There are major elements of risk to be considered in developing a data communications strategy for the ITC system. Networks of a few hundred personal computers are considered to be state-of-the-art; we must plan for 5000 to 10000. Further, these computers will be used by people with a wide variety of skills, ambitions, and motivations in a relatively uncontrolled environment. Perhaps the most serious risk of all is that the underlying technology for the local area network is new and untried in a production system. Because of these risks, it is essential that no unnecessary risks be taken in the logical design for communications software. The strategy with lowest risk is to build software on a coherent, well-established architecture that has been used successfully in other real products and systems.

Invention will be required for many aspects of the ITC system, including communications. There exists, however, a good architectural base for designing the ITC data communications facilities -- IBM's Systems Network Architecture (SNA). SNA is defined precisely and is maintained by a large (over 100 professionals) IBM organization. SNA and ready access to support from the SNA architects is an important technological resource for the ITC.

The basic design strategy is to implement Systems Network Architecture (SNA) where SNA provides solutions for required function. Where additional function is required, the ITC will work with the IBM Architecture and Telecommunications (A&T) organization in Raleigh to define SNA enhancements. The major example of additional function needed for the ITC is distributed control functions (e.g., directory services, configuration services and network management). Work is already underway in A&T to address these requirements. When ITC schedules require us to implement without a completed architecture, every effort will be made to stay within the framework and general principles of SNA.

3.2 LOGICAL DESIGN

Communication facilities in the ITC system will provide for general interprocess communication independent of physical location and machine boundaries. The structure of these facilities can be described as forming a number of logical networks. These logical networks share the physical resources of the system (storage for data and re-entrant programs, bandwidth of the local area network, and processor cycles). The physical configuration of the system need not be apparent to the processes.

Informally, a logical network consists of two or more concurrent processes that are able to exchange information and are co-operating to accomplish some task. The processes may be fully interconnected (Figure 1) or have some hierarchical structure in the communication paths(Figure 2). For example, the components of the ITC file system will probably form a logical network with the structure shown in Figure 3. If this logical network is mapped onto one possibility for a physical configuration, the result is shown in Figure 4. Another example from the ITC system is a logical network of "mail servers" which may be interconnected with the file-server logical network as shown in Figure 5.

In order to fully explore the expected structure of the ITC communications, a number of likely logical networks are briefly discussed in the following paragraphs. The distinguishing feature of these logical networks is that a number of processes, normally executing on physically separate processors, must communicate to provide a particular service to the users of the ITC system.

File Services Network

The processes in the file services network cooperate to provide safe storage of user files and reliable access to them. This service will eventually provide page-at-a-time access and swapping services. File services processes will exist in work stations, cluster service machines, and central service machines.

Mail Services Network

The processes in the mail services network cooperate to provide delivery of electronic mail (ranging in size from a few bytes to large documents) among work stations. Mail services processes will exist in work stations, cluster service machines, and central services machines.

Print Services Network

The processes in the print services network cooperate to provide formatting transformations and actual printing of documents. Print services processes will exist in work stations and service machines.

Directory Services Network

The processes in the directory services network cooperate to provide name assignment, name resolution, and attribute access for named objects in the system. Directory services processes will exist in work stations, cluster service machines and central service machines.

Authorization & Authentication Services Network

The processes in the authorization and authentication network cooperate to provide control of capabilities granted to users of the system. These processes will exist in work stations and service machines.

Network Management Network

The processes for network management cooperate to provide centralized control for network operations. Included in these functions is collection of error information, error correlation and problem determination, configuration management, remote system control, and performance monitoring. Network management processes will exist in work stations, cluster servers, and central service machines.

3.3 AN SNA-BASED LOGICAL DESIGN

We describe the facilities defined in SNA that are the building blocks for these logical networks in the following sections.

3.3.1 SYNCHRONOUS COMMUNICATIONS

The most basic service provides synchronous (end-to-end) communication between pairs of concurrent processes. Processes communicate through named "ports" called logical units (LU). LUs are pair-wise connected by abstract links called sessions. In order for a session to be established and maintained, all physical elements (transmission media, storage and cycles at intermediate and end nodes) must be available simultaneously.

The particular LU and session type in SNA that provides generalized interprocess communication is LU_6.2. Using LU_6.2, processes can be written so that interprocess communication is done in the same way without regard to location of the processes (same address space, different address spaces, different machines). This makes it possible to move processes about in the network without changing them.

An LU_6.2 may have sessions established with many different LUs of the same type ("multiple sessions") and many sessions with each ("parallel sessions"). A session is the basic mechanism for synchronizing exchanges between a pair of processes and thus can be used for only one process-to-process exchange (called a "conversation") at a time. An LU may be used by many independent processes and thus the LU is capable of serially allocating a session to processes requesting a conversation. A process may use many different LUs but can be engaged in only one conversation with a particular other process at a time (see Figure 6). LU_6.2 services can be used to construct a logical network of ITC proc-

esses (for example the file services network) as shown in Figure 7. The network formed by all interconnections of LUs is called the SNA transport network.

The services of LU_6.2 are defined in the IBM publication Transaction Programmer's Reference Manual for LU Type 6.2 (publication GC30-3084). These services are defined in terms of semantics for a set of programming language elements ("verbs"). These semantics can be implemented in the ITC programming environemnt as a set of procedure calls. The collection of "verbs" is commonly called the "LU_6.2 application programming interface (LU_6.2 API)". A brief review of them is given below.

3.3.1.1 LU_6.2 Application Program Interface

The LU 6.2 API is a structured interface. It is defined in terms of formatted functions, or verbs, and the rules for their usage. The rules are the allowed sequences in which a process may issue the verbs. Process execution, in terms of the verbs, occurs when a process issues a verb and the LU executes it; verbs are issued and executed one at a time. When a process issues a verb, its execution is suspended while the LU executes the verb.

From a process standpoint, only the conversation is meaningful and all verbs operate in the context of a conversation. The activation of the LU-LU session and the actual messages the LUs exchange on that session are hidden from processes. It should also be noted that either or both of the processes in a conversation could also be engaged in conversations with other processes. However, any pair of processes can only be engaged in a single conversation at a time.

This section presents an overview of the verbs. They are divided into three groups, with each group representing a subset of LU 6.2 services. Thus, the LU 6.2 API is actually composed of the three distinct groups of verbs. The three groups are:

Conversation verbs
Mapped conversation verbs
Control operator verbs

The mapped conversation verbs are slight generalizations of the conversation verbs and do not appear useful in the ITC system. The control operator verbs are intended for use by operator-related programs; i.e. programs which assist a network operator in performing the functions related to the control of an LU 6.2. At some later time, a description of them will be included here.

Basic LU 6.2 services do not require implementation of all of the conversation verbs, parameters, and return codes. Rather, these have been organized into sets. The base set consists of those conversation verbs,

parameters, and return codes that every LU 6.2 must have. The several option sets, however, may or may not be individuallly implemented as needed.

The base and option sets are further categorized in terms of local and remote support. Local support is the support of verbs, parameters, and return codes provided by a LU at the local end of a conversation, as seen by the local process. Remote support is the support of verbs, parameters, and return codes provided by the LU at the remote end of a conversation, as seen by the local process. This distinction is important because only certain verbs and parameters require any processing at the remote end of a conversation; others are processed entirely at the local end.

The conversation verbs are intended for use in interprocess communications. The complete list of verbs included in this protocol boundary follow (those marked by *' are required in the base set.)

*ALLOCATE - Allocates a conversation connecting the process which issues the verb with another process. This verb must be issued prior to any verbs referring to the conversation.

BACKOUT - Restores all protected resources allocated to the process to their status as of the last synchronization point. Protected resources are those which are protected by the synch-point service of LU_6.2.

*CONFIRM - Sends a confirmation request to the other process and waits for a reply. This allows the two processes to synchronize their processing. The execution of the FLUSH verb is included as part of the execution of this verb.

*CONFIRMED - Sends a confirmation reply to the other process. This is issued in response to the receipt of a CONFIRM request sent by the other process.

*DEALLOCATE - Deallocates a conversation from the process. The process issues this verb when it is finished using the conversation. The execution of the CONFIRM or FLUSH verbs may be included as part of the processing of this verb.

FLUSH - This causes the LU to transmit all the information it has buffered for the conversation. The buffered information results from ALLOCATE, DEALLOCATE, PREPARE_TO_RECEIVE, SEND_DATA, and SEND_ERROR verbs previously issued by the process.

*GET_ATTRIBUTES - This returns information pertaining to the conversation. Examples of the information which may be requested are conversation's mode name, the name of the LU associated with the other process,and the conversation's synchronization level.

GET_TYPE - Returns the conversation's type, either conversation or mapped-conversation. This is useful when the other process allocated the conversation and the issuing process needs to know which group of verbs to use.

PREPARE_TO_RECEIVE - Changes the conversation from send state to receive state, in preparation for receiving information from the other process. A SEND indication is sent to the other process and the state of its conversation changes from receive to send. The execution of the CONFIRM or FLUSH verbs may be included as part of the processing of this verb.

*RECEIVE_AND_WAIT - Waits for information to be received from the other process and then receives the information. If information is already available, it is received without waiting. The information can be data, conversation status, or a request for confirmation or synch point. Control is returned to the issuing process with an indication of the type of information which was received. If the conversation is in send state when this verb is issued, the execution of the PREPARE_TO_RECEIVE and FLUSH verbs is included as part of the processing of this verb.

*REQUEST_TO_SEND - Notifies the other process that the issuing process would like to change the state of its conversation from receive to send. The state of the conversation will be changed when the issuing process subsequently receives the send indication from the other process (as a result of its issuance of the PREPARE_TO_RECEIVE verb).

*SEND_DATA - Sends data to the other process. The data is in logical records, with each logical record consisting of a two byte length field followed by a variable length data field. The length field contains the length of the entire record, which is limited to a maximum of 32767 bytes. The amount of data sent with a particular issuance of the SEND_DATA verb is independent of the size of the logical record which is being sent; i.e. a partial logical record, or multiple logical records may be sent.

*SEND_ERROR - Informs the other process that the issuing process has detected an error. For example, the process can issue this verb to truncate a logical record it is sending, to inform the other process of an error it has detected in data it has received or to reject a confirmation request. Upon the succesful execution of this verb, the issuing process is in the send state, and the other process is in the receive state.

SYNCPT - Advances all protected resources allocated to the process to the next synchronization point. Protected resources are those which are protected by the synch-point service of LU 6.2.

The following ten option sets have been defined for the conversation verbs:

1) The allocation of a conversation between two local processes connected through the same LU. This appears to provide an optimization for interprocess communication in the local environment.

2) SNA-defined modename SNASVCMG. The less said about this, the better.

3) Delayed allocation of a conversation. Normally, when a process issues the ALLOCATE verb, control is not returned to it

until the conversation has been allocated. If the allocation is delayed for any reason, the process must wait. This option set allows control to be returned to the process immediately; the LU will continue to attempt to allocate the conversation, and will notify the process when it subsequently issues another verb.

4) Immediate allocation of a conversation. Normally, when a process issues the ALLOCATE verb, control is not returned to it until the conversation has been allocated. If the allocation is delayed for any reason, the process must wait. This option set allows control to be returned to the process immediately; if the conversation could not be allocated, the LU simply informs the process and does not continue to attempt to allocate the conversation.

5) This option set implements the synch-point services of LU 6.2. It includes support for both the BACKOUT and SYNCPT verbs.

6) This option set allows a process to record error information in the system's error log.

7) Support for the FLUSH verb.

8) Support for the GET_TYPE verb.

9) Support for the PREPARE_TO_RECEIVE verb.

10) This option set allows a process to issue a PREPARE_TO_RECEIVE verb that includes the function of the CONFIRM verb, and specify that control is not to be returned to it until after information in addition to the CONFIRMED reply is received from the other process. Option set 9 is a prerequisite for this option set.

3.3.2 ASYNCHRONOUS COMMUNICATION

In many logical networks, synchronous communication with full connectivity from end to end is either not required or very difficult to achieve. For example, a computer electronic mail system that required the sender and all receivers to be simultaneously present would be essentially worthless. In this and many other cases, asynchronous or delayed delivery is much more desirable.

The architecture for asynchronous delivery of interprocess communications is embodied in the Store and Forward (SAF) services of SNA. Store and Forward services are provided by a logical network (called the SAF network) of architected processes called SAF Service Units (SSU). SSUs are named elements of the SAF network. A process that requires delivery of information to one or more destination processes can submit the request to a SSU using a defined set of "verbs" (procedure calls) called the "SAF

application programming interface (SAF API)" (see Figure 8). The information delivered can range from a few bytes of control or state data to an arbitrary set of files.

Once the SSU at the origination point has received the request, the SAF logical network is responsible for ultimate delivery to the destination(s) or for reporting failure if necessary. The SSUs in the SAF network use architected protocols for establishing routes, safe storage of information at intermediate points, reporting progress, and notifying receivers of a delivery.

The logical network of SAF Service Units internally uses the synchronous interprocess facilities of the LU_6.2 transport network. LU_6.2 sessions and conversations are dynamically established as needed to satisfy delivery requests. SAF also requires an interface to the file services system for obtaining safe storage of information. (see Figure 9). A brief review of the semantics of the SAF API is given below.

3.3.2.1 SAF Services Application Programming Interface

Like the LU_6.2 API, the SAF Services API is described in terms of procedure calls ("verbs") that can be issued by processes to request SAF services. IBM documents define the SAF services architecture in detail.

SEND_DISTRIBUTE
The origin process issues SEND_DISTRIBUTE to initiate asynchronous delivery or asynchronous feedback. Control is returned to the issuing process when the request parameters have been checked for syntax validity and the request has been accepted by SAF. The parameters of this verb provide destination name(s), identification of the data (files) to be delivered, service level required (e.g., "fast", "secure"), variables for status reporting, information for receiver notification, etc.

RECEIVE_DISTRIBUTE
The destination process issues RECEIVE_DISTRIBUTE to take delivery of information from SAF services. The parameters of this verb are used to return information about the delivery (e.g., origin, type, identification of the data (files), status, etc.).

UPDATE_DIRECTORY
A process issues UPDATE_DIRECTORY to change information in SAF directories of named origin or destination entities in the SAF network. The parameters of this verb indicate the type of update (ADD, CHANGE, DELETE, REPLACE) and the new information.

GET_SSU_ATTRIBUTES
A process uses this verb to determine the name of the SAF Services Unit (SSU) providing services to that process.

3.3.3 PHYSICAL UNIT SERVICES

In SNA network nodes there is a component that manages physical resources, especially the activation and deactivation of communications links, and reports status needed for network management. This component is called the physical unit (PU). Physical units are classified according to the role of the node in an SNA network. For example, the physical unit in a "peripheral" node (one with no routing or network address translation functions) is a type 2 PU. The particular PU type that will be implemented in all ITC work stations and service machines is PU Type 2.1 (PU_T2.1). Processes using SNA services are not explicitly aware of the PU and no programming interface is defined.

3.3.4 CONNECTION NETWORK ARCHITECTURE AND INTERFACES

The local area network will be a token passing ring, as described in the proposal document 802.5 submitted to the IEEE standards committee. This architecture defines an end-to-end physical transmission subsystem and the formats and protocols necessary to manage the subsystem. The subsystem is characterized by a single address space. Any number of logical point-to-point connections among elements of this address space can share the bandwidth of the physical transmission medium. Protocols that control the shared medium are called Medium Access Control (MAC).

The ITC physical network will be constructed from components of the IBM local area network products including media, active and passive wiring concentrators, and hardware adapters to attach work stations and service machines to the media. The adapters implement the MAC protocols and provide a register (MMIO) and DMA interface to the SNA layers executing on the attaching machine. Examples of functions at the interface to software consists of: Adapter DMA, Adapter to System Interrupts, System to Adapter Interrupts, Adapter MMIO Commands, Command and Status Processing, and Adapter Initialization. Also defined is logical "control block" interface that defines how data frames and control information are organized for processing by the adapter.

3.3.5 ARCHITECTED PROCESS MODELS

In addition to the basic services for synchronous and asynchronous interprocess communication, the SNA framework encompasses architected processes ("models") that provide designs for essential service elements of the network. Architected process models use either the LU_6.2 API or the SAF API for communication. The process models that appear necessary (or useful) for the ITC system are described briefly below. Except for the communications network management (CNM) model, these process models are

more closely related to other parts of the ITC system (e.g., file and mail services) than to the transport network, and are more appropriately discussed elsewhere. They are mentioned here for completeness in describing the SNA framework. IBM documents give definitions for these architectures.

3.3.5.1 Distributed Communication Network Management (CNM)

Distributed CNM allows a hierarchically organized set of processes to manage networks of SNA nodes. This includes such functions as problem determination, threshold monitoring, statistics gathering, traces, and tests.

3.3.5.2 Directory Services Architecture (DSA)

Directory services architecture specifies the protocols, object definitions, and algorithms necessary to provide a hierarchy of directories distributed throughout a network. To the user of the directory services, the appearance is of a single network-wide directory. Functions provided include generating unique names, name "resolution" (associating information,e.g.,location, with names) and dynamic maintenance of directories as objects enter, leave, or move about in the network.

3.3.5.3 Distributed Data Model (DDM)

Basically, DDM is an architecture for managing and accessing data in a network of heterogeneous systems. Data is defined to include everything from file models through database models, both hierarchal and relational. It also includes data structures from expert systems and speech and visual data representations.

3.3.5.4 Document Interchange Architecture (DIA)

DIA is intended to provide a means by which distributed office-application processes interchange documents. Examples of its functions are: DISTRIBUTE a document, RETRIEVE a document, or SEARCH for a document. It provides a general mechanism by which requests and documents may be interchanged among a wide range of machines.

3.3.6 STRUCTURE OF SNA SERVICES IN ITC MACHINES

Basically, each ITC work station and service machine will contain (as an integral component of its execution environment) an SNA node implementation. The internal structure and algorithms of an SNA node are defined in the IBM publication SNA Format and Protocol Reference Manual: Architectural Logic (publication SC30-3112). The SNA-node component of work stations and service machines will differ only in the types of architected process models that are provided. Each will have the basic services of PU_T2.1, LU_6.2, and SSUs utilizing the IBM local area network.

3.3.6.1 Service Machines

Figure 10 gives the internal structure likely to be used in an arbitrary service machine that supports all ITC logical networks.

3.3.6.2 ITC Integrated Work Stations

See Figure 10.

3.3.6.3 ITC Low Function Work Stations

To be supplied

3.3.6.4 Non-ITC Work Stations

To be supplied

3.3.7 DATA STREAM ARCHITECTURE

The final element required for meaningful communication is a model for interpreting and acting on the content of messages exchanged. This is particularly true in the case of documents that are prepared on one workstation and are destined to be formatted for display or printing on a variety of devices. The SNA framework also encompasses a number of architected data streams that define parsing and interpretation rules for

information display. These architectures fall in two broad catagories:(a) editable (revisable) data streams and (b) final-form data streams. These architected data streams are more closely related to other parts of the ITC system (e.g., print services, user interfaces) than to the transport network, and are more appropriately discussed elsewhere. They are mentioned here for completeness in describing the SNA framework. IBM documents define these data streams in detail.

3.3.7.1 Editable Data Streams

3.3.7.1.1 Script

3.3.7.1.2 Office Information Interchange Architecture - Level 3

3.3.7.2 Final Form Data Streams

3.3.7.2.1 Composed Page Data Stream

3.3.7.2.2 Document Architecture - Level 2

3.4 GATEWAYS TO NON-SNA NETWORKS

The computing environment at C-MU currently consists primarily of a large complex of computers (six DEC 20s) in the Computation Center, and an equally large complex of computers (three DEC 10s and thirty DEC VAXs) in the Computer Science Dept. The Computation Center serves the needs of the general C-MU population, while the Computer Science Dept. computers serve the needs of that department's members.

A DECnet and an Ethernet respectively link together the computers of the two complexes. They also each have connections to different long distance networks. The Computer Science Dept., for example, has a connection to the ARPANET.

It is necessary to provide some manner of connection between the ITC network and these two complexes. This is required in order to provide users with an uninterrupted access to services and programs on these computers before the services and programs are migrated, if indeed they ever are, to the ITC network. These network interconnections are also necessary to allow users to move their files to the ITC network, and begin using it.

Several possibilities for accomplishing this have been explored. They range from building ASCII terminal emulation code for the work stations to

building an internetwork server. The internetwork server would take advantage of existing hardware, which allows the connection of a 370 channel to a DEC Unibus and protocol conversion code written by DEC for the VAX, which converts between the DECnet and the SNA LU_2 protocols (this assumes that it will be possible for us to build a conversion between SNA LU_2 and LU_6.2 protocols). If it is decided to provide a direct connection to the ARPANET, it will also be necessary to build an internetwork server to convert between the TCP/IP and the SNA LU_6.2 protocols.

3.5 NETWORK MANAGEMENT

High availability is essential in the ITC system and, therefore, there must exist comprehensive network management tools to measure the health of the local area network components, report noteworthy incidents or conditions, and recover from failures. The key elements of network management will be (1) a centralized network operations focal point ("help desk"), and (2) distributed monitoring services that gather incident information and forward it to the central focal point. The programs that support network management run as processes in the distributed network management network.

The IBM product for distributed monitoring of the local area network resides in an IBM PC attached to the network and communicates with other network management processes in the network (e.g., those at a central focal point). All network management processes should use the architecture for distributed communication network management (based on LU_6.2 and SAF services). Use of existing IBM products for the centralized network operator interface should be investigated carefully. The candidates are NCCF (which requires VTAM), NPDA, and NLDM (which require NCCF).

4.0 IMPLEMENTATION PLAN

4.1 SCOPE

This implementation plan for ITC communication is expressed in terms of the functions and facilities provided by the network in September, 1984. At that date there will be an operational system with adequate performance and reliability to support useful work on the attached work stations and service machines. The physical connection layer will utilize the IBM strategic local area network products. The communications network will provide support on 9/84 for >200 work stations and > 3 service machines, including a central facility (4341). The implementation code for this support should be of high quality, suitable for enhancement with functions needed for full deployment of the ITC system.

4.2 FOUNDATIONS

The implementation plan for the ITC communications system is based on the following observations.

(1) A "starter set" of SNA services will be needed for all ITC work stations and service machines. The first version of this package should support LU_6.2 for interprocess communication, PU_T2.1 for physical unit services, and the local area network. A second release should add (as an option) SAF services for asynchronous communications.

(2) A common programming language and execution environment is planned for the ITC work station and service machines. This commonality should be exploited by developing a portable implementation of the SNA services that can be moved to the various ITC machines with a minimum of tailoring and optimization.

(3) A model implementation of basic SNA services, including LU_6.2, PU_T2.1, and SAF, has been developed by the IBM architecture group. This model is written in a high-level language (Format and Protocol Language - FAPL). An implementation of LU_6.2 is also available from IBM-Yorktown in an even higher-level language, NIL (Network Implementation Language). An early prototype of the SNA services can be obtained by manually translating these model implementations into the common programming language and execution environment used in ITC machines (almost certainly optimizations will be required later to achieve satisfactory performance). IBM product implementations also exist in a variety of languages and programming environments.

(4) The implementation plan can be based on a strategy of implementing first those components where the SNA definition is complete and for which

model implementations already exist (PU_T2.1, LU_6.2, SAF). In parallel, we can work to have the other needed parts of SNA completed by early 1984. Thus 1983 can be devoted to developing basic SNA services for work stations and service machines and making these components operational using the local area network. Activity in 1984 should emphasize implementation of distributed network services (e.g., network management, directories, authorization & authentication), migration aids(e.g, gateways to CMU systems), and instrumentation.

4.3 FUNCTIONS:

The key components of the network in September, 1984 are:

- Full PU_T2.1 implemented for work stations and service machines
- Base LU_6.2 for work stations and service machines
- LU_6.2 option sets implemented as needed
- SAF Service Units implemented for work stations and service machines
- Terminal emulation for access to TOPS-20 systems
- File Transfer Program (FTP) from TOPS-20 to ITC file servers
- Distributed network services
  - Directory with dynamic maintenance (i.e. support for adding/removing names of network objects)
  - Network Management for session-level integrity, problem determination, and integration with local area network management aids.
- Instrumentation for performance data

4.4 MILESTONES:

- 06/83 "Thin layer" LU_6.2 to support work station and service machine development work using available transmission media
- 10/83 Full PU_T2.1 and base LU_6.2, using available transmission media
- 01/84 IBM local area network support
- 03/84 SAF Services

- 04/84 Distributed directory services
- 05/84 Terminal emulation and FTP for TOPS-20 (Gateway)
- 06/84 Distributed network management (with local area network)
- 07/84 Network instrumentation

4.5 ASSUMPTION:

Common programming language and execution environment for both work stations and service machines

4.6 DEPENDENCIES:

- IBM local area network prototypes available 09/83
- Architecture for distributed functions available 01/84
- "Thin layer" LU_6.2 depends on 03/83 selection of interim work station, service machine, and transmission media and protocol

Physical connection dependencies for testing transport network

- 04/83 work station <-----> work station
- 05/83 work station <-----> Service Machine
- 05/83 Service Machine <-----> Service Machine
- 08/83 Service Machine <-----> 4341
- 10/83 IBM local area network <-----> work station
- 10/83 IBM local area network <-----> Service Machine
- 01/84 IBM local area network <-----> 4341

Physical connection dependencies for development environment

- 03/83 work station <----> 4341
- 03/83 Service Machine <-----> 4341