NIST Special Publication 500-176

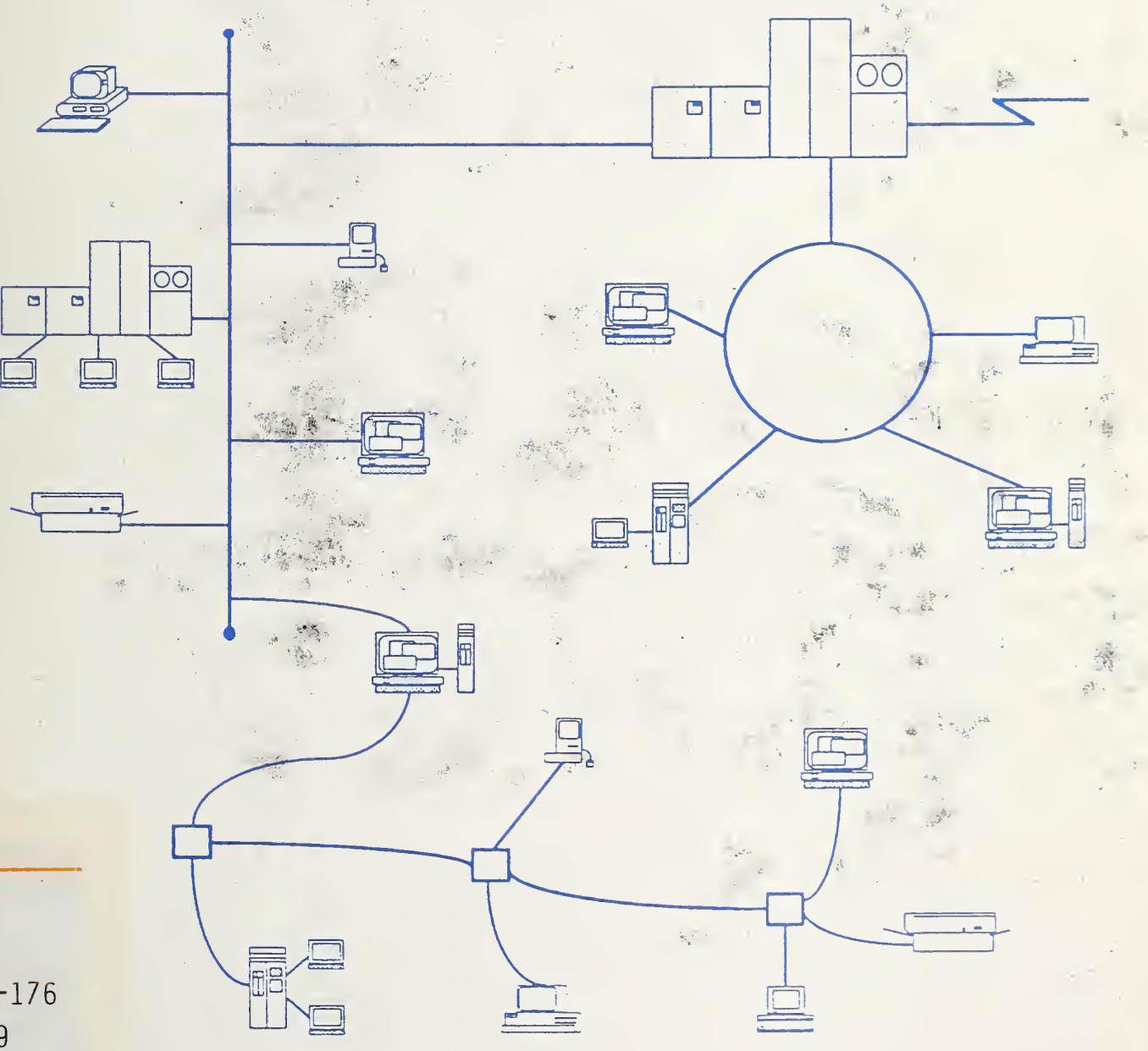
Computer Systems Technology

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

NIST

Introduction to Heterogeneous Computing Environments

John F. Barkley
Karen Olsen



NIST Special Publication 500-176

Introduction to Heterogeneous Computing Environments

John F. Barkley
Karen Olsen

National Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

November 1989

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
Raymond G. Kammer, Acting Director

NIST

Reports on Computer Systems Technology

The National Institute of Standards and Technology (NIST) (formerly the National Bureau of Standards) has a unique responsibility for computer systems technology within the Federal government. NIST's National Computer Systems Laboratory (NCSL) develops standards and guidelines, provides technical assistance, and conducts research for computers and related telecommunications systems to achieve more effective utilization of Federal information technology resources. NCSL's responsibilities include development of technical, management, physical, and administrative standards and guidelines for the cost-effective security and privacy of sensitive unclassified information processed in Federal computers. NCSL assists agencies in developing security plans and in improving computer security awareness training. This Special Publication 500 series reports NCSL research and guidelines to Federal agencies as well as to organizations in industry, government, and academia.

Library of Congress Catalog Card Number: 89-600784
National Institute of Standards and Technology Special Publication 500-176
Natl. Inst. Stand. Technol. Spec. Publ. 500-176, 37 pages (Nov. 1989)
CODEN: NSPUE2

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1989

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402

ABSTRACT

Computer networks are becoming larger not only in the number of nodes connected but also in the geographic area spanned. In addition, networks are becoming more diverse in the variety of equipment from which the network is implemented. This report provides an introduction to the concept of a *heterogeneous computing environment.* It characterizes heterogeneous computing environments from the point of view of the generic services provided. Standards are necessary in order to implement heterogeneous computing environments. The report provides an introduction by example to the types of technical standards that are necessary in a heterogeneous computing environment and illustrates how such standards can be used to provide services.

Contents

List of Tables

List of Figures

1 Introduction

Computer networks are becoming larger and more diverse. Networks are becoming larger not only in the number of nodes connected but also in geographic areas spanned. Networks are becoming more diverse in the variety of equipment from which the network is implemented. This equipment includes not only the communication equipment but also the systems which make up the nodes of the network. The communication equipment includes local area networks of different transmission media (e.g., CSMA/CD and token ring) interconnected by means of bridges, routers, and gateways to form wide area networks. The nodes may be, for example, mainframes, minicomputers, workstations, and/or personal computers. Components of the communication equipment and the nodes of the network can be provided by many different producers. Such a communication network is referred to as a *heterogeneous computing environment.* Nodes connected to such a network become part of this heterogeneous computing environment.

On the cover of this report, there is an illustration of a local site which is part of a heterogeneous computing environment. The installation is made up of three different local area networks each with a different transmission medium and each consisting of personal computers, workstations, and minicomputers/mainframes with terminals. The three local area networks are interconnected with routers. One large computer connects the local area networks at this site into a wide area network made up of many such local sites.

This report[1] provides an introduction to heterogeneous computing environments, such as that depicted on the cover, by:

1. Characterizing heterogeneous computing environments from the point of view of the generic services provided.

2. Presenting some examples of standards that are currently used to provide these generic services.

This report is intended for managers and users in government and industry to assist in their evaluation, management, and use of computer networks. This introduction discusses how applications are evolving to make increasing demands on the communication capabilities of a heterogeneous computing environment. Section 2 characterizes heterogeneous computing environments by the services they provide in support of more sophisticated applications. Section 3 describes some examples of communication, application programming interface, and user interface standards needed to support a heterogeneous computing environment. The information in section 3 is provided for the more technically advanced reader.

[1]Because of the nature of this report, it is necessary to mention vendors and commercial products. The presence or absence of a particular trade name product does not imply criticism or endorsement by the National Institute of Standards and Technology, nor does it imply that the products identified are necessarily the best available.

There are many standards which are necessary in order for a heterogeneous computing environment to be able to function. For example, the Government Open Systems Interconnection Profile (GOSIP, FIPS[2] 146) and the Applications Portability Profile (POSIX, FIPS 151-1, Appendix A) are available to assist Federal Agencies and other organizations in acquiring and using equipment within heterogeneous computing environments. It is not the goal of this report to list all of the formal and defacto standards which can apply to a heterogeneous computing environment, nor to recommend or imply that the standards presented here as examples are the best choices. The standards used as examples in this report are among those currently in widespread use. In the future, this may not be the case. Section 3 provides an introduction by example to the kinds of standards that are necessary in a heterogeneous computing environment and illustrates how such standards can be used to provide services.

Heterogeneous computing environments have evolved in response to the ever increasing need for applications to be able to communicate more easily and effectively. For example, consider how two applications, spreadsheet and electronic publishing, are becoming more sophisticated in their use of communications.

When spreadsheets first appeared as an application, they ran exclusively on personal computers. Spreadsheet data was shared by exchanging floppies. As personal computers became nodes on networks, spreadsheet data was shared by transferring files between personal computers and then integrating them into the local copy. Sharing spreadsheet data required much hands-on operation by the user.

The spreadsheet applications now being developed permit this data sharing to occur without the user being forced to leave the keyboard. In many cases, sharing spreadsheet data can happen without the user being aware that sharing is taking place. Users interacting with spreadsheet applications often need not know whether the data resides locally or remotely. The spreadsheet application fills in cells with data automatically obtained remotely from other text, spreadsheet, and database information.

Electronic publishing has emerged as an application much more recently than spreadsheets. Electronic publishing currently is used in the following manner. Small personal computers are used to create parts of large documents. These parts are transferred to sophisticated graphics workstations which assemble and finish the document in both draft and final versions. Since the personal computers and workstations are usually on a network, the transfer of text and graphics can be done on the network. However, this often involves several operational steps on the part of users in order to coordinate their efforts.

Electronic publishing applications now being developed allow for a less labor intensive effort in the cooperative development of large documents. In a manner similar to new spreadsheet applications, the finishing workstations can automatically access the text and graphics pieces of large documents over the network. Moreover, users of the small personal computers can generate draft copies of the completed document from their keyboards so

[2]Federal Information Processing Standard.

they can see how their parts fit into the whole. These draft copies can be automatically assembled and printed remotely.

In order to support the demand for more flexible and transparent communications in applications such as spreadsheet and electronic publishing, large communication systems of diverse hardware and software components have emerged. In the past, sophisticated computer communication systems and the applications which ran on them came from a single producer, i.e., they were *homogeneous* computing environments. For better or worse, this is no longer the case. Complex computer communication systems and their applications come from many different producers. Thus, the name *heterogeneous* computing environment was coined.

The emergence of heterogeneous computing environments results in organizations having a larger number of communication equipment producers from which network components can be acquired. In many cases, this larger community of producers results in both a lower total network cost and/or a greater network functionality. While many organizations still choose a homogeneous computing environment, for Federal Agencies, this choice is rarely available because of the requirements for full and open competition in acquisitions.

To describe a heterogeneous computing environment as a collection of systems and components from different producers is not very informative. It is useful to know what a heterogeneous computing environment can do for a user, whether that user is an application developer or an end user. In the next section, heterogeneous computing environments are characterized from the point of view of the generic services they provide.

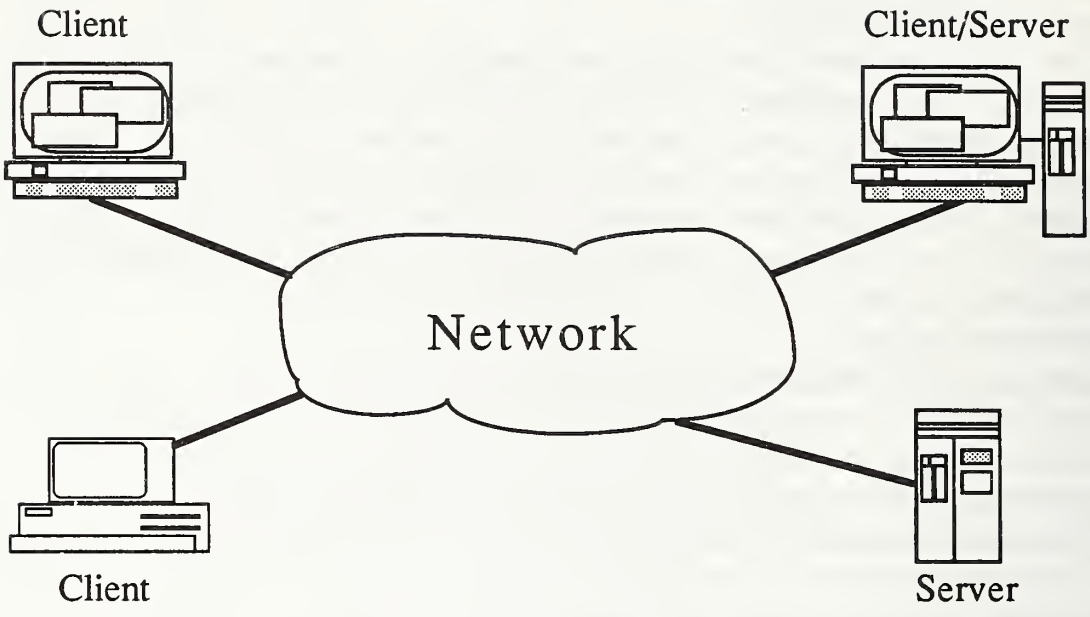


Figure 1. Client/Server Model.

2 Heterogeneous Computing Environments

A client/server model is used to describe the generic services provided in a heterogeneous computing environment. While other models may also apply, client/server models serve well as a means of describing the services outlined in this section.

In a client/server model, a node in a heterogeneous computing environment (e.g., personal computer, workstation, minicomputer, mainframe) may be characterized either as a client, a server, or a client/server. A *client* node provides no services to any other node. A client is only able to be the recipient of services provided by other nodes. A node that provides services is a *server*. Often a server can also be a client. Such a node is referred to as a *client/server*. The term *client* may also refer to a process which runs on a client node or to the person on whose behalf a client process is acting. Similarly, the term *server* may also refer to a process which runs on a server node. In this report, the term *client* refers to a *client node* and the term *server* refers to a *server node*.

In figure 1, a personal computer and a workstation are clients. They provide no services to any other node, but they may be the recipient of a service, such as mounting a file system or directory, from either the server or client/server. The client/server may also be the recipient of services.

Originally, nodes of computer networks were exclusively mainframes and minicomputers. The user accessed the network only by means of a terminal connected to one of the nodes. The services provided by the network were usually login (sometimes called *vir-*

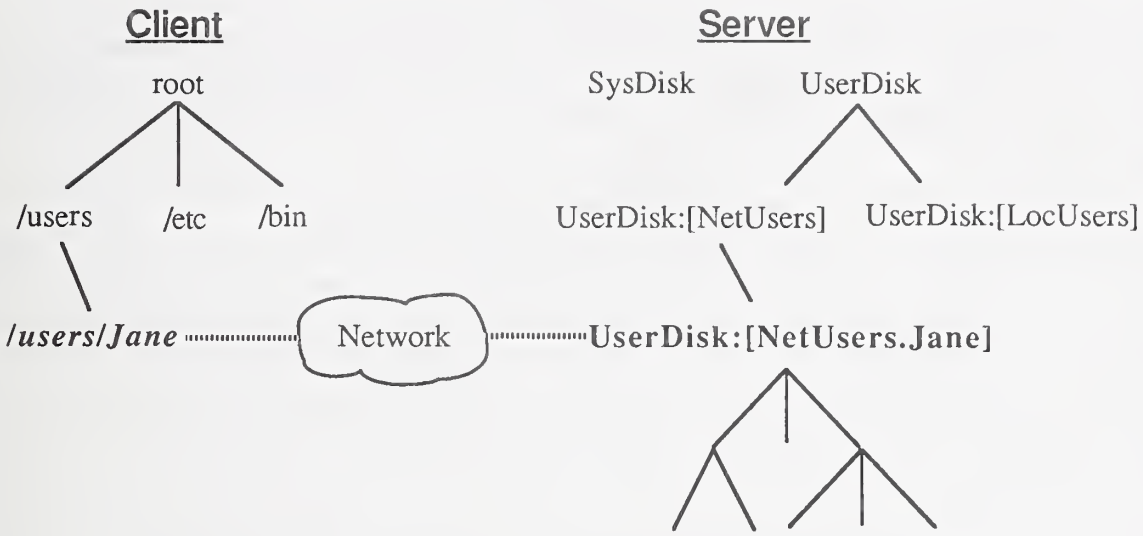


Figure 2. Client access to remote mass storage on server.

tual terminal or ***remote login***), file transfer, and mail. Each node of the network was a client/server for these three services. Users were provided with an interactive terminal session with any node on the network. They were able to transfer files between any two nodes. Finally, they were able to send mail to any node and receive mail from any node.

With the advent of the personal computer, networks were implemented whose nodes were exclusively small computers. Such a ***personal computer*** network provided file service and mail. Each node of a personal computer network was usually either a client or a server and typically, there were no client/server nodes. There was no login service provided since each personal computer ran applications locally. Although applications may be downloaded from a server, their primary function was to provide file service for data (i.e., transparent access to data on mass storage). File transfer between clients was accomplished by copying files to and from a server. Additionally, servers sometimes acted as a post office for electronic mail. Clients accessed the mail server to send and receive mail.

This distinction between large computer networks and personal computer networks has somewhat disappeared. In a heterogeneous computing environment, the services of the large computer network and the personal computer network are integrated into three basic types of services which are available to all sizes of clients from all sizes of servers or client/servers. In a heterogeneous computing environment, clients, servers, and client/servers may be, for example, mainframes, minicomputers, workstations, or personal computers. Most services can be grouped into three major categories: distributed file system, distributed computing, and messaging.

2.1 Distributed File System

The concept of a *distributed file system* is the generalization of the concept of file service. File service provides a client with transparent access to part of the mass storage of a remote server. Transparent access means that, to the user on a client, the remote mass storage of the server is available on the client as though it were mass storage locally connected to the client.

Figure 2 illustrates the concept of transparent access. The "/" notation is used to indicate the location of a file within the directory tree of the client. The ":" and "." notation is used to indicate the location of a file on the server (see sec. 2.1.1). The client has, as part of its file system, a part of the file system of a remote system which is providing file service. The directory tree which begins at */users/Jane* on the clients directory tree physically resides on the server's directory tree at the point *UserDisk:[NetUsers.Jane]* which is part of the server's mass storage device *UserDisk*. Note that, in this example, although the syntax of file reference differs between the client and the server, the client uses its file reference syntax (i.e., the "/" notation) for all file references including those which physically reside on the server. The fact that the client and server use different file reference syntax notations implies that the client and server have different operating systems. In a heterogeneous computing environment, a client may have access to several servers simultaneously. In addition, on some servers, a client may have transparent access to peripheral devices on servers other than mass storage, such as, printers and modems.

Since a client and server may be from different producers, a user on a client is provided with transparent access to remote peripherals in the user interface paradigm of the client, not the server. This means that, once remote mass storage is attached, users issue the same commands (e.g., copy, rename, delete) to access remote mass storage as they do to access local mass storage. Similarly, once a remote printer is attached, the commands to print on the remote printer are the same as the commands to print on the local printer. The file system/device model (i.e., how file or device references are supported in the command language) used by each client is the access provided by servers to clients in a heterogeneous computing environment.

2.1.1 File System/Device Models

Most clients have one of two types of file system/device models: single-tree and device-based. The single-tree model is one in which all physical devices are part of a single directory tree (see fig. 3). The directory trees of each mass storage device are attached to a node in the system's single directory tree. A file reference in the command language is a path in the single directory tree, such as, */users/jane/file2*. Physical devices other than mass storage are also attached to the single directory tree. Thus, a printer is simply a file reference in the single directory tree. In figure 3, */dev/xd*, */dev/tty1*, and */dev/printer* represent a disk, a terminal, and a printer. The directories */bin* and */etc* respectively contain system executable files and system configuration files.

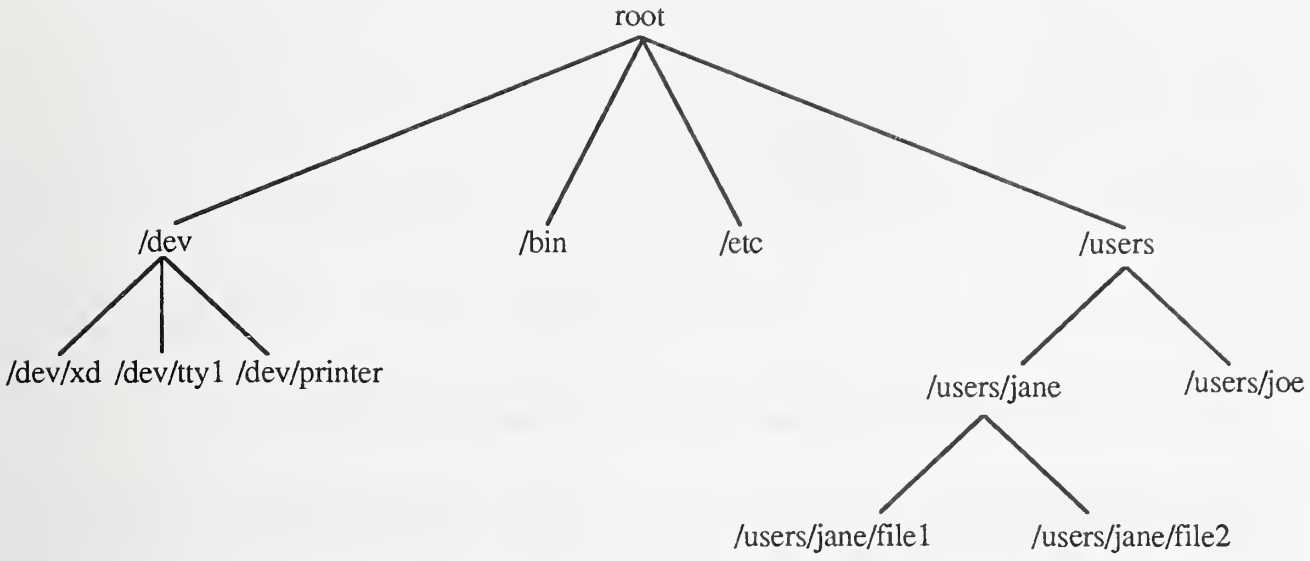


Figure 3. Single-Tree File System/Device Model.

The device-based model is one in which each physical device must be specifically referenced (see fig. 4). Each mass storage device, such as *SysDisk* and *UserDisk*, has its own collection of files. A file reference includes the name of the physical device and the name of the file. If the file system is hierarchical (i.e., supports directory trees), then a file reference includes the name of the physical device and the path in the directory tree on that device. For the file reference *UserDisk:[jane.file2]*, *UserDisk* is the name of the physical device, *jane* is the name of the directory, and *file2* is the name of the file. Devices other than mass storage simply have names and are treated in the command language in a manner specific to the type of device. For example, a printer device would have commands in the command language specific to printers like *print* and *interrogate print queue*.

2.1.2 Types of Clients

There are three types of clients in a heterogeneous computing environment: diskless, dataless, and datashare. A diskless client has no mass storage of its own. All file access is provided by the distributed file system. A diskless client boots, loads applications, and stores its data on servers. A dataless client has only enough local mass storage to enable it to boot and perhaps, load applications. All of its data is kept on servers. A datashare client can function more autonomously. It has some applications and some data stored locally but, at the same time, may have some data and applications stored remotely. Figure 2 shows a dataless client which uses a single-tree file system/device model which is attached to a server which uses a device based file system/device model. The files in */users/Jane* on the client appear to be local but actually reside on the server.

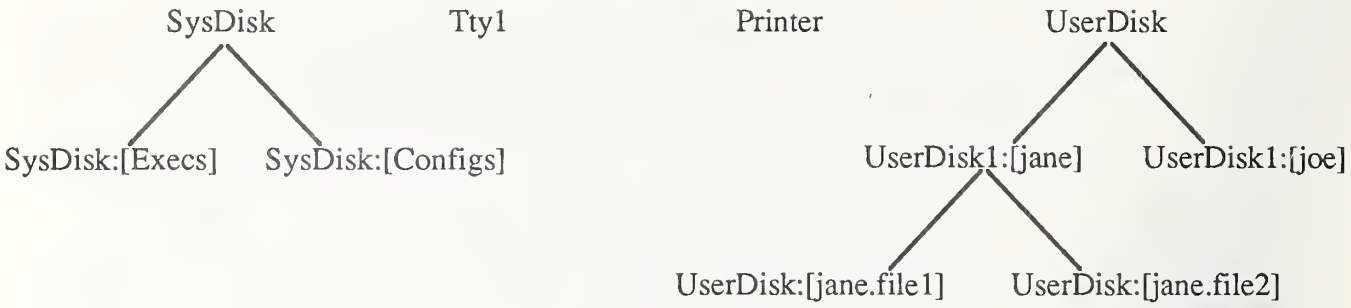


Figure 4. Device Based File System/Device Model.

Although the distributed file system in a heterogeneous computing environment is available over a large geographic area, as a practical matter, clients typically attach to servers which are close geographically, such as, within the same building or building complex. This is because most long distance network links operate at much slower speeds (e.g., 1.5 Megabits) than short distance links (e.g., 10 Megabits). The convenience of a distributed file system is diminished by slow speed communication.

A common solution to this problem is first to perform a file transfer over low speed links to local (i.e., geographically close) servers where clients have higher speed access. Then, clients may have transparent access at suitable speeds. For the purpose of file transfer, files are referenced in the file system/device model of the server rather than the client. A file reference is usually of the form: (*network-node-name*, *file-reference*).

2.2 Distributed Computing

The second major category of service in a heterogeneous computing environment is *distributed computing*. Distributed computing refers to the concept of running an application or applications on remote node(s). Distributed computing is not the downloading of an application to be run locally. Such downloading is a service of the distributed file system. Distributed computing is the execution of an application on a remote server or servers with the results transmitted over the network to the client. Distributed computing takes four forms: login, remote execution, transaction processing, and cooperative processing.

2.2.1 Login

Login is the service which provides a user on a client with an interactive session on a server. By means of the login service, the client behaves as though it were a terminal directly connected to a host where the host is a login server on the network. Typically, the login session is conducted using the user interface of the login server. Some clients

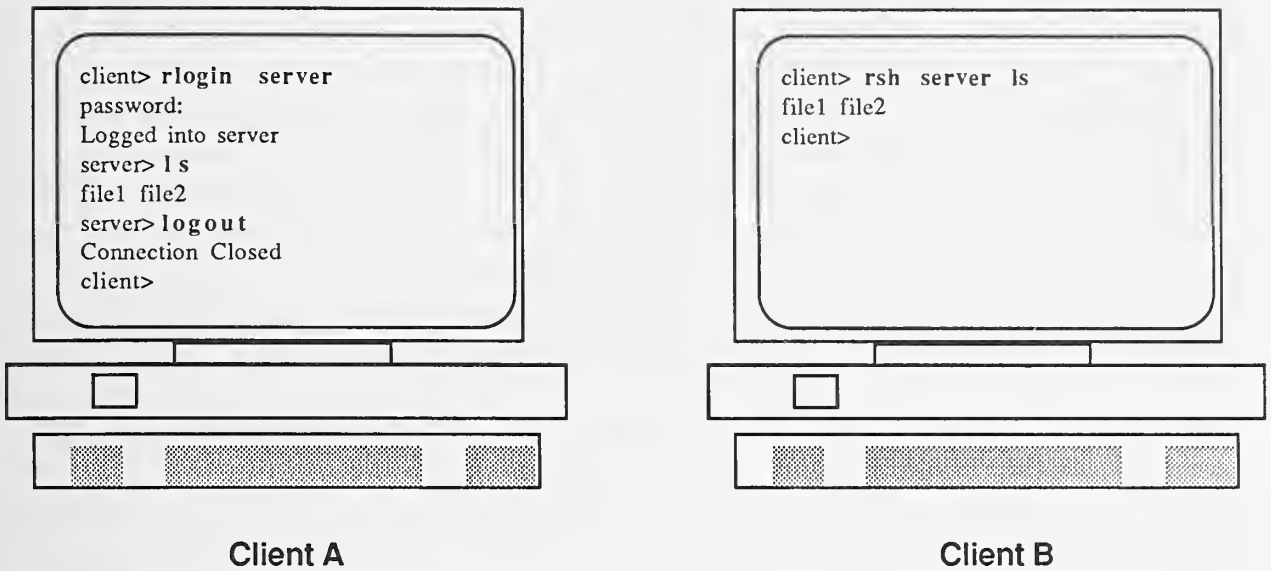


Figure 5. Login (Client A) vs. Remote Execution (Client B).

are able to have several login sessions to multiple servers active simultaneously. Normally, each login session requires a dedicated process on the client and a dedicated process on the server. For those clients with mouse/window user interfaces, each login session is usually a separate window. If the client is also a server, then remote interactive applications can access local mass storage for data by means of the distributed file system. If not, then file transfer to a server may be used to allow access to data. Login permits access to remote applications which are highly interactive in nature.

Figure 5 shows the interaction between a user on client A and a login server in the typical login session. The purpose of the session is to obtain a directory listing of a directory on the login server. The **bold** indicates what the user enters.

In the future, the need for a login service will diminish. Users will interact with applications as though the application were resident locally, whereas, in fact, the processing of the application may be taking place on several different nodes (see sec. 2.2.4).

2.2.2 Remote Execution

Remote execution is the service which provides a user on a client access to remote applications which are not highly interactive in nature. Remote execution is differentiated from login in several ways. With login, the user on the client interacts with each server in the command language (i.e., user interface paradigm) of the server. With remote execution, the user issues commands to the server using the user interface of the client, i.e., commands

executed by the server are issued in the same language as commands to the client. From an implementation point of view, a login session requires a process on the client and a process on the server dedicated to each interactive session throughout its establishment. On the other hand, remote execution normally only requires the client to execute a single process in order to access multiple servers at the same time. Moreover, a single process on the server is usually sufficient to manage processing requests from all clients.

Figure 5 shows the interaction between a user on client **B** and a remote execution server. As is the case with the user on client **A** accessing a login server, the user on client **B** is obtaining a directory listing from a remote system. The **bold** indicates what the user enters. Note that the user on client **B** is not *logged into* the server, but is simply entering a command which happens to be carried out on the server. In many cases, remote execution happens without the user's knowledge. With remote execution, all operations can appear as though they are happening locally even when they take place remotely.

Remote execution is also different from the venerable concept of *remote job entry.* Remote job entry is the remote initiation of batch processing on a host. Remote execution on a server normally proceeds at interactive priorities (i.e., the same priorities as interactive sessions). Batch priorities are usually less than interactive priorities. In addition, the concept of remote execution includes the concept of applications on different nodes closely interact with each other in a highly synchronized manner.

In the future, the need for remote execution services will diminish. The functionality of remote execution (as shown in the example of fig. 5) will be replaced by applications that access remote nodes on behalf of, and unbeknownst to the user (see sec. 2.2.4).

2.2.3 Transaction Processing

The concept of *transaction processing* services evolved from the requirements of database applications. A *transaction* is an update (i.e., change) made to information in a database. Many transactions involve a query as well. The database is queried, a decision made on the basis of the information obtained from the query, and then a change made to the database. An application maintaining a database on a server in a heterogeneous computing environment must be capable of managing transactions, which arrive virtually simultaneously from clients, in a manner which insures the integrity of the information.

In a heterogeneous computing environment, transaction processing services are provided in order to meet the special needs of distributed database applications. A transaction applied against a database on a server can cause the server to generate transactions against databases on other servers, and so on. The initial transaction generates a flow of transactions which may be depicted as a flow through a tree structure. Each leaf of the tree represents a server running an application maintaining a database. Transaction processing services help the server database application control and synchronize these transaction flows so that the information contained in all of the databases is reliable and up-to-date.

2.2.4 Cooperative Processing

The availability of remote execution services and transaction processing services has given rise to the concept of a *distributed application.* A distributed application is an application in which the interaction between the user and the application appears as though all of the processing is taking place locally, but, in fact, the processing may be distributed across several nodes of the network. The most common example of a distributed application can be found in a distributed database application. The user on a client initiates queries to a database whose data is distributed among several nodes of the network. To the user, the data appears to be local. A server (or servers) accepts the queries, generates the information, and sends the results back to the client. Implementations of spreadsheet and electronic publishing applications now becoming available (as described in sec. 1) are also examples of distributed applications.

Distributed applications make use of *cooperative processing* services. The application programmer uses cooperative processing services for processing required by the application and, in some cases, need not be aware of the location where the processing is taking place. The cooperative processing service may be able to determine automatically where processing is to take place. Cooperative processing services include the functionality provided by both remote execution and transaction processing services. In the future, the concepts of cooperative processing and distributed applications should replace the concepts of login, remote execution, and transaction processing. Users will only interact with distributed applications that are implemented using cooperative processing services.

2.3 Messaging

Messaging services are associated with mail and conferencing applications. Electronic mail has been available on computer networks from their beginning. Mail servers in a heterogeneous computing environment act like local post offices. Users on a client must normally obtain their mail remotely from the server to which it was sent, but may send mail directly. Only client/servers can both send and receive mail locally.

Conferencing services allow messages to be sent and received immediately. Conferencing in a heterogeneous computing environment is similar in concept to using a telephone. While mail messages can be transferred and stored without the active participation of the users, conferencing messages are lost if the receiver does not take immediate action. A notification that a message was received is displayed on a user's screen and if the response is not forthcoming, the message is lost. This form of message passing can be just between two users (like a telephone call) or among several users (like a telephone conference call). In most cases, users on clients can use conferencing directly without the support of any servers.

2.4 Standards

The heterogeneous computing environment is supported by standards. Three major kinds of standards are necessary in a heterogeneous computing environment: communication protocols, application programming interfaces, and user interfaces. Standard communication protocols are necessary so that systems from different producers can connect to a communication network and understand the information being transmitted. Standard communication protocols result in interoperability between diverse computer systems. Standard application programming interfaces are necessary so that software developers can implement applications whose source code runs almost unchanged on systems from different producers. Standard application programming interfaces result in portable applications. Standard user interfaces are necessary so that users are able to interact with remote applications on systems from different producers in the same manner as they interact with applications locally. Standard user interfaces result in *portable* users, i.e., users who can easily access applications regardless of the hardware platform on which the applications are run.

In section 3, several standards which are currently used as a means of implementing some of the services described in this section are presented. The standards discussed in section 3 are communication protocols and/or application programming interfaces.

3 Examples of Standards for Heterogeneous Computing Environments

In this section, several standards needed by heterogeneous computing environments and already in use in network implementations are examined. Such standards include those necessary in order to implement the services described in section 2 and can be categorized as either communication protocol standards and/or application programming interface standards. This section is provided for the more technically advanced reader.

Implementations of the services described in section 2 require certain lower level capabilities. For example:

- In order to implement a distributed file system for a client, a set of file manipulation primitives must be defined. These file manipulation primitives would be essentially the ones common to local file systems (e.g., read, write, delete, rename). Moreover, they would be passed as messages between clients and servers and would be accessible to applications through procedure calls.

- In order to implement remote execution and transaction processing, processes on different nodes must be able to coordinate their activities almost as though they were processes on the same system. This implies that there must be a standard set of process control and interprocess communication primitives. Just as is the case with file manipulation primitives, process control and communication primitives would have two forms: messages and procedure calls.

- In a heterogeneous computing environment, there is no commonality of data representation. Even systems from the same producer may have dissimilar data formats. In addition, different programming languages have different ways of representing simple data types and aggregate data types. If file systems are to be shared among heterogeneous systems and if processes on heterogeneous systems are to communicate, then there must be a common data representation between systems and languages.

- Beyond a standard data representation, heterogeneous computing environments must provide some means of user identification and authentication. In a distributed file system, servers must identify users in order to invoke file protection mechanisms. For remote execution and transaction processing services, servers must be able to invoke processes which can act on resources (e.g., the distributed file system) in the name of a user.

Required lower level capabilities such as those listed above are provided by standards. Network File System, External Data Representation, Remote Procedure Call, Advanced Program to Program Communication, and X Window System are examples of such standards. They fit into the International Standards Organization (ISO) Basic Reference Model

Table 1. Location of example protocols in ISO Basic Reference Model

Application	Presentation	Session	Presentation/Session
Network File System X Window System	External Data Representation	Remote Procedure Call	Advanced Program to Program Communication

as indicated in table 1. However, they are not ISO standards. None of the standards used as examples in this section are formal standards. The standards in this section are defacto standards that are widely used and which are either being considered by various standards organizations for inclusion in a formal standard or have already formed the basis for a standard which is under development by a standards organization.

3.1 Network File System

The Network File System (NFS), initially developed by Sun Microsystems, is a communication protocol and application programming interface which is emerging as a defacto standard for distributed file system services in a heterogeneous computing environment. It permits a partition of a server's file system to be associated with either a device or a subdirectory on a client depending on the file/device model of the client's file system (see sec. 2.1.2 and fig. 2). Although NFS was first implemented within a Unix environment, NFS is now implemented within several different operating system environments. File manipulation primitives supported by NFS include: read, write, create a file or directory, remove a file or directory, lookup file name. NFS includes an Application Layer protocol and is usually part of a Transmission Control Protocol/Internet Protocol (TCP/IP) protocol stack.

NFS is referred to as a *stateless* system. This means that the server does not maintain the state of files, from the client's point of view, in file systems mounted by clients. There are no *open* or *close* primitives in NFS. Each file manipulation request from the client contains all of the information necessary for the server to complete the request. The server responds fully to every client's request without being aware of the conditions under which the client is making the request. Thus, for example, if the server fails, the client may just continue to request file access until the server is able to respond. Only the client knows the state of a file for which service is requested. In a system where a server maintains the states of files as they are accessed by each client, the failure of a client, a server, or the network is difficult to recover from in an acceptable manner that will restore the states of clients and servers to the conditions in place before the failure.

However, the absence of knowledge on the part of the server concerning what clients are doing to files can lead to unpleasant consequences. For example, one client may have a file open on a server and another client may delete the open file. The server is unaware that a client has the file open. In particular, a fully stateless mechanism cannot be used in database applications. Record and file locking inherently involves managing the current state of a file by a server.

In order to permit database applications, the record locking mechanism specified in the System V Interface Definition (SVID) is supported by another protocol, the Network Lock Manager, which works in conjunction with NFS. The Network Lock Manager uses *Status monitors*, daemon processes on both clients and servers, to initiate recovery procedures in the event of failure. By means of status monitors, clients and servers notify each other concerning their operational state. If a client fails, then when the client is restarted, the server removes all lock information for that client and the client resubmits all lock requests. If a server fails, then when the server is restarted, it notifies all clients and clients resubmit their lock requests.

Most NFS implementations use Remote Procedure Call. Such implementations usually support the user authentication methods of Remote Procedure Call discussed in section 3.3.

Another example of a communication protocol and application programming interface for distributed file systems is Remote File System (RFS) developed by AT&T. However, RFS only supports a distributed file system among nodes which have Unix compatible file systems.

RFS is an example of an approach to distributed file systems which is termed *stateful*, i.e., the server maintains information about the state of the file on the client, such as, whether the file is open. This is necessary in order for RFS to support the full Unix file system semantics. The file systems of many operating systems do not support the semantics of a Unix file system. Participation in a distributed file system implemented using NFS does not require the semantics of a Unix file system. Thus, file systems from many different producers are able to be part of a distributed file system implementation using NFS.

Since RFS supports the full semantics of a Unix file system, the application programming interface for RFS is the Unix input/output (I/O) application programming interface. Since NFS is intended to support operating systems other than Unix, NFS can be described as having two layers of application programming interface. The high layer is the I/O application programming interface of the client operating system. The low layer is the NFS Remote Procedure Calls which provide direct access to the NFS file manipulation primatives. Typically, the NFS client's high layer application programming interface, which provides the file system semantics of the client's operating system, is implemented using the low layer interface. For example, if the client operating system is Unix, then the Unix I/O application programming interface would also provide access to NFS file systems on servers.

However, because NFS is a *stateless* systems, i.e., the server does not maintain infor-

mation about the state of a file on a client, NFS, by itself, is unable to support the full file system semantics of many operating systems. Thus, when an application applies the I/O application programming interface of the client's operating system, only some of the client's file system semantics may be supported. NFS is neither a specification of file system semantics for operating systems nor a specification of an I/O application programming interface for operating systems. NFS is a specification of a network protocol, a set of file manipulation primatives, and an application programming interface to those primatives.

The Institute of Electrical and Electronics Engineers (IEEE) Standards Committee P1003, which developed the POSIX standard, is developing a standard file semantics for file access over networks. This standard is referred to as POSIX Transparent File Access (TFA). The TFA standard specifies file semantics for file access between clients and servers over networks. In addition, TFA will specify an application programming interface, an enhancement of the POSIX application programming interface, which may be used by applications on clients of TFA file systems regardless of the native operating system of the client. The TFA standard will specify file semantics which are operating system independent and capable of being supported by both *stateless* systems, such as NFS, and *stateful* systems, such as RFS. The TFA committee is using the POSIX standard, which specifies file semantics for standalone systems, as the starting point for its efforts. However, conformance to the TFA standard will not require conformance to the POSIX standard.

Another example of a protocol related to the manipulation of files over a network is the protocol defined by the ISO File Transfer, Access, and Management (FTAM). FTAM is a *stateful* system. FTAM was initially developed as an Application Layer protocol for the transfer of files from one network node to another. Although FTAM has been used to implement distributed file systems, FTAM currently has no protocol mechanisms for the manipulation of directories on file servers. Each distributed file system implementation using FTAM must develop its own mechanism for dealing with directories. Currently, FTAM includes a protocol for manipulating files across a network while NFS is a protocol for manipulating file systems across a network. However, FTAM is now being modified to include directory manipulation capabilities.

3.2 External Data Representation

External Data Representation (XDR) is an encoding of simple and aggregate data types enabling the exchange of information between different systems and programming languages. The simple and aggregate data types specified by XDR resemble those in the C programming language. In order to transmit data between nodes, a data stream is translated from the internal data format of the sending node to the XDR encoding. The XDR encoded data are sent over the network to the receiving node which translates the data from the XDR encoding to its native representation. XDR resides in the Presentation Layer and is usually part of a TCP/IP protocol stack. XDR was initially developed by Sun Microsystems, whose implementation includes an application programming interface.

XDR uses implicit typing of data in its encoding. This means that the receiving node may have to have a priori knowledge of the sequence of data types in the stream transmitted by the sending node. Information as to what data types are in the stream is not necessarily part of the encoding, i.e., the stream consists of only the encoded data and not the type information for the data. The type information for the data stream may be included in the stream (i.e., explicit typing) but how typing is included in a data stream must be agreed upon by the sending and receiving nodes. A methodology for explicit typing is not part of the XDR standard.

The Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules for ASN.1 ISO standards are another approach to data representation. The acceptance of these standards, from the point of view of their use in implementations, is increasing and is expected to dominate in the future. ASN.1 requires that explicit typing be used in data streams and specifies the encoding for the explicit typing. The disadvantage of explicit typing is that the data streams are necessarily longer because of the inclusion of the typing information. Moreover, added processing is needed by the receiving side to interpret the typing information in order to translate the encoded data. In many cases, communicating processes already know what data types to send and receive. The extra space in messages and the extra processing required for explicit typing is not required. However, in the future, with the advent of more powerful processors and higher speed networks, these disadvantages will become less of a concern.

3.3 Remote Procedure Call

Remote Procedure Call (RPC) is a model that specifies how cooperating processes on different nodes in a heterogeneous computing environment can communicate and coordinate activities. RPC is an approach to providing distributed computing services in a heterogeneous computing environment. The paradigm of RPC is based on the concept of a procedure call in a higher level programming language. The semantics of RPC are almost identical to the semantics of the traditional procedure call. The major difference is that while a normal procedure call takes place between procedures of a single process in the same memory space on a single system, RPC takes place between processes on clients and servers in a heterogeneous computing environment.

3.3.1 Model

Figure 6 illustrates the basic operation of RPC. A process on a client issues a normal procedure call to a *client stub*. The client stub is a library routine which accepts the call parameters from the calling procedure, creates a message containing the call parameters, and calls the transport layer interface. The transport layer interface transmits the message with the call parameters to the server transport layer interface. The server transport layer interface issues a call to the *server stub* which takes the call parameters from the message

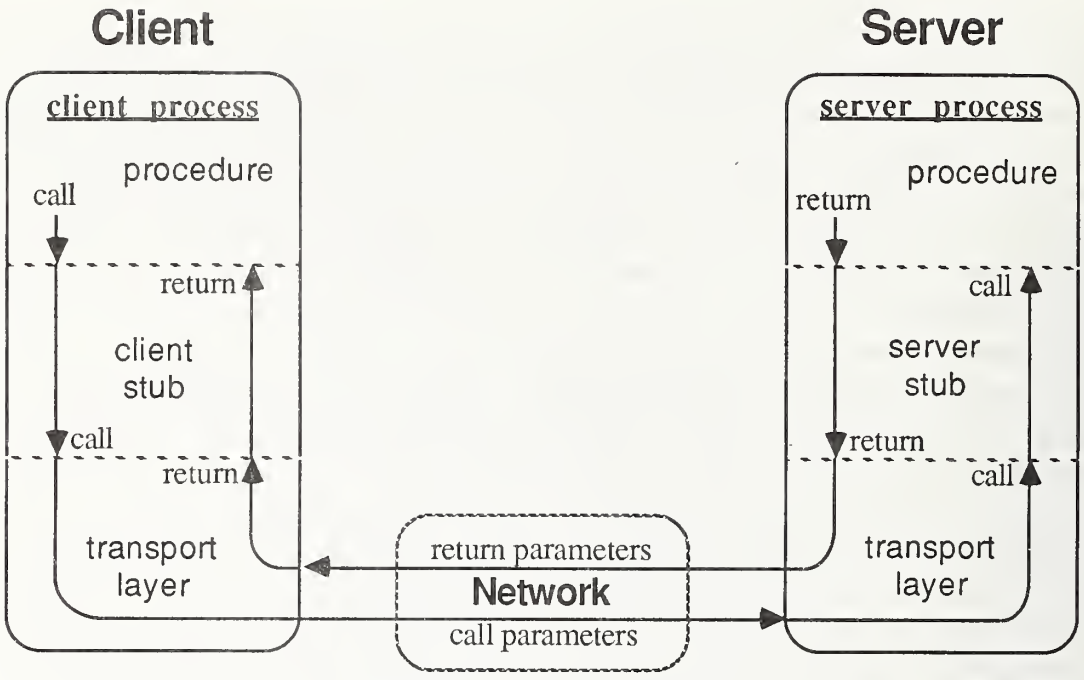


Figure 6. RPC model.

and calls the procedure which does the processing.

When the server process procedure has completed, it returns to the server stub with the return parameters. The server stub encapsulates the return parameters into a message which is passed to the transport layer of the server. The server transport layer sends on the network the return parameter message to the transport layer of the client which passes the message to the client stub. Finally, the client stub extracts the return parameters from the message and returns them to the calling procedure in the required form.

Like a normal procedure call, RPC is a synchronous operation, i.e., the client process is blocked until processing by the server is complete. This is not acceptable for many applications. As a consequence, the RPC model is enhanced to include the concept of a *lightweight process*. A lightweight process (also known as a *thread*) is an independent execution path within a normal process. A normal process can consist of several lightweight processes, each behaving like a normal process from the point of view of CPU usage. However, all lightweight processes of the same process share the same address space. Thus, context switches between lightweight processes may be done more economically than context switches between normal processes. A client process can initiate a RPC in a lightweight process and then proceed with other processing. The completion of the RPC can then be indicated either by a status check or a software interrupt.

Note that the RPC model does not explicitly deal with the problem of systems which have different data representations. However, almost all implementations provide a mechanism to solve the problem of different data representations.

3.3.2 Implementation

Several producers have implemented the RPC model. Some make use of the same communication protocol and have similar application programming interfaces. Many others have announced their intention of implementing the RPC model. In order to provide a more in-depth description of the functionality of RPC, this section describes the RPC implementation from Sun Microsystems. In this section, the term *RPC* refers to this implementation. RPC is the primary mechanism used in the implementation of NFS (see sec. 3.1). RPC is a session layer protocol which is part of a TCP/IP protocol stack. It uses either TCP or UDP as a transport layer protocol. In addition, RPC uses XDR as its data representation (see sec. 3.2).

The RPC implementation includes an application programming interface which has three layers. An increasing amount of control over the functioning of the RPC mechanisms is provided from the highest application programming interface layer to the lowest layer. The highest layer provides the least amount of control. The lowest layer provides the maximun amount of control.

The highest layer is the boundary between the client procedure and the client stub (see fig. 6). Library routines (e.g., **rnusers**, which returns the number of users on a remote node) are used by client applications just as they would use any other library procedures. The client application need not be aware that it is using RPC.

The intermediate layer of the RPC application programming interface is used to implement RPC client and servers for specific applications. The intermediate layer can be used to develop client and server stubs and consists of the three routines: **registerrpc**, **svc_run**, and **callrpc**. The routine **registerrpc** identifies to the server the server process and the procedures that the server process is to handle. The server process is identified to the server by two numbers: the program number and the version number. When the server process is started, it identifies itself using **registerrpc** and then calls **svc_run**, which places the server process in a suspended state waiting to be called. From this point on, when a client initiates a call to one of the server procedures using **callrpc**, the server activates the selected server process which calls the selected procedure. The client identifies the remote procedure by specifying in the call to **callrpc** the server node name, the program number, the version number and the procedure number. Server process program numbers must be unique on the network.

The intermediate layer is suitable only for single applications. It does not permit user authentication, nor allow timeouts to be specified, nor permit a choice of transport mechanisms (i.e., TCP or UDP). The lowest layer must be used in order for these and other parameters to be controlled. The lowest layer consists of several routines which permit fine control over the operation of client and server stubs. Both the intermediate and lowest layers of the RPC implementation use XDR for data representation. The XDR library has many routines for the translation of both simple and aggregate data types into and out of XDR. In addition, the routines in the library may be used to develop new routines to

translate aggregate data types for specific applications.

The implementation of client stubs and server processes is facilitated by the use of **rpcgen**, a program generator. **rpcgen** receives as input specifications of client and server procedures along with declarations of the data types to be represented in XDR and generates C code for client stubs and server processes. Although **rpcgen** does not permit the fine control of RPC parameters that is possible when developing directly using the lowest layer of the application programming interface, **rpcgen** generates code which uses the lowest layer. The generated code may then be modified to meet specific needs.

Lightweight processes are available to be used in conjunction with RPC. The lightweight process library has many routines for the creation and removal of lightweight processes, for communication and synchronization between lightweight processes, for optional user scheduling of lightweight processes, and for optional user management of lightweight process stacks. Lightweight processes exist within a single Unix process and all lightweight process scheduling takes place within the CPU time allocated for the Unix process. Lightweight processes belonging to different Unix processes do not compete directly with each other for CPU time. Because the lightweight processes are not implemented in the Unix kernel but by means of library routines, developers using lightweight processes must be aware that a system call within a lightweight process may block all of the lightweight processes within the single Unix process.

The RPC implementation is capable of supporting several different methods of user identification and authentication. There are four kinds of user authentication: none, two types of Unix style authentication, and DES authentication, a technique which uses the Data Encryption Standard[3] (DES). A client can request RPC service and supply no identification. This enables the server to provide services without the overhead of user authentication. For many applications, e.g., read-only access to files, this may be appropriate.

Unix style authentication involves identifying a user to a server by means of the Unix user ID and group ID. The client's user must have an account on the RPC server. Each service request from the client is accompanied by the user's Unix style identification. Having identified the user, the server is able to process the request according to the access to which the user is entitled. Unix style authentication is implemented in two ways. In the first method, each RPC request includes the user's user ID and group ID. Upon receipt of the request, the server must check the password file. The second method eliminates the checking of the password file for each request. When the first request is made, the server returns a short identification structure. Subsequent requests may now include only the short identification structure. The server keeps a table which permits quick access to a user's original credentials using the short identification structure.

The fourth type of authentication is DES authentication. This approach fixes two shortcomings of Unix style authentication:

1. The use of Unix style identification may not be meaningful to a non-Unix node.

[3]Data Encryption Standard, FIPS 46-1.

2. The lack of some verification mechanism to help insure that the user making the request is really that user.

With DES authentication, each user is given a network name which is unique network-wide. To this name is applied a verification procedure by the server each time a request is made. With the first request, the user includes a DES key, called the *conversation key*, which is used in subsequent requests. The client uses the key to DES encrypt the time and include this encrypted timestamp in subsequent service requests. The server decrypts the timestamp using the conversation key. If the timestamp is greater than the one seen on the previous request and within a selected window of the server's time, the user identification is verified and the request processed.

3.4 Advanced Program to Program Communication

Database applications are an important part of a heterogeneous computing environment. Consequently, the transaction processing services which support database applications (see sec. 2.2.3) should be available in heterogeneous computing environments. Advanced Program to Program Communication (APPC) is a communication protocol and application programming interface standard initially developed by IBM in order to provide transaction processing services. However, APPC is now used as a general purpose mechanism for providing distributed computing services in a heterogeneous computing environment. Like NFS, APPC has been implemented within several operating system environments including Unix. APPC is a protocol in the Presentation and Session Layers. APPC was used as input for the ISO Transaction Processing Standard which is now a Draft International Standard (DIS). The ISO Transaction Processing Standard is very similar to the communication protocol of APPC except that the ISO standard permits full duplex communication links. However, the ISO standard does not specify an application programming interface.

3.4.1 Example

APPC provides primitives, called verbs, which enable procedures, called transaction programs and network application programs, to communicate and synchronize activities with each other over a network. The basic functioning of APPC can be understood by looking at a sample communication session between two transaction programs as summarized in figure 7. The example demonstrates how a query and response is carried out in APPC. Network nodes in APPC are referred to as *logical units* (lu). Communication between transaction programs (tp) running on logical units in APPC is accomplished in a *conversation*, i.e., a grouped set of messages passed between procedures. The example of figure 7 illustrates a conversation between **tp(a)** on **lu(a)** and **tp(b)** on **lu(b)**.

tp(a) on **lu(a)** initiates the conversation by means of the ALLOCATE verb. **tp(a)** is requesting **lu(b)** to activate **tp(b)**. **tp(a)** then sends a long query, too long to be sent by one call to SEND_DATA because the query is longer than **lu(a)**'s internal buffer. The

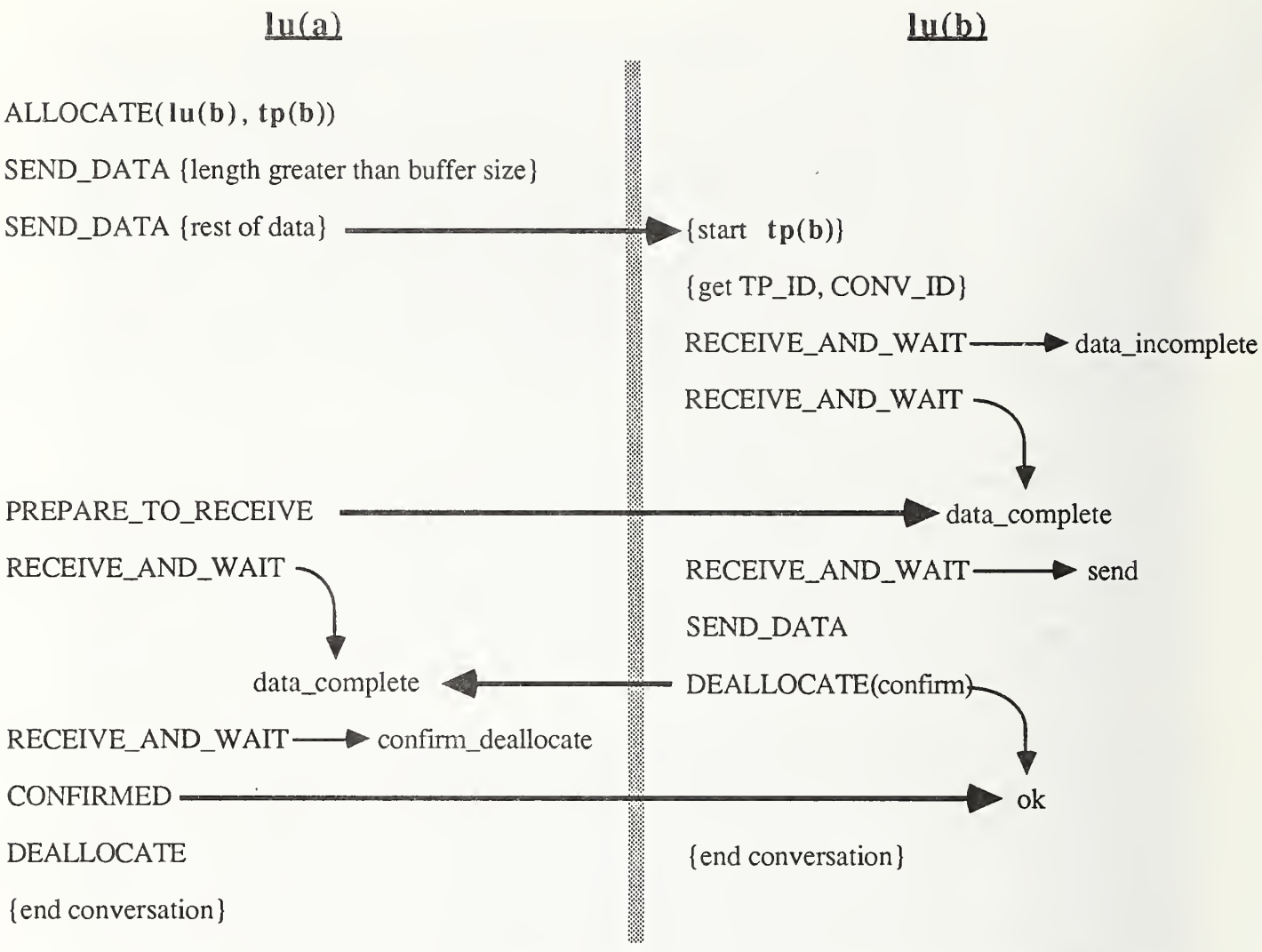


Figure 7. Example of query/response conversation in APPC.

arrows crossing the vertical center line representing the network in figure 7 indicate when a message is actually sent on the network. The smaller arrows indicate a status indication returned to a transaction program from the call to the verb. Note that nothing is sent on the network until a buffer is filled as a result of the second SEND_DATA call. Procedures on logical units must be aware of the buffering mechanism in APPC so that they can be sure that a message was sent and not hang waiting for a response.

At this point, lu(b) has received the ALLOCATE request and the first part of the query. The rest of the query remains in a buffer on lu(a). tp(a)'s second SEND_DATA started a new buffer but did not fill it. The transaction program tp(b) is started and obtains its transaction program ID (TP_ID) and conversation ID (CONV_ID), which it needs as parameters for conversation verbs. tp(b) now reads the message by using RECEIVE_AND_WAIT. The first RECEIVE_AND_WAIT returns a status of data_incomplete.

Therefore, **tp(b)** issues a second RECEIVE_AND_WAIT which causes **tp(b)** to block until the rest of the query arrives.

Meanwhile, **tp(a)** wants to be able to receive messages from **tp(b)**. APPC is a half-duplex protocol, i.e., the communication link between **tp(a)** and **tp(b)** is uni-directional. Only one side of the conversation has permission to send data at any one time. When a conversation begins, the transaction program which initiates the conversation has the send permission. PREPARE_TO_RECEIVE is used by **tp(a)** to pass the send permission to **tp(b)** so that **tp(b)** can respond to the query. At that time, the partially filled buffer in **lu(a)** containing the rest of the query is sent to **lu(b)** along with a message passing the send permission. **tp(b)** is unblocked and receives a status indication that the complete query has been received. **tp(b)** initiates another RECEIVE_AND_WAIT and receives a status indication that it has received send permission. **tp(b)** sends the response back to **tp(a)** and terminates the conversation by means of the DEALLOCATE verb. **tp(b)** uses the *confirm* parameter to DEALLOCATE which causes the conversation to terminate only if **tp(a)** answers that the response was received.

Meanwhile, **tp(a)** was blocked on a RECEIVE_AND_WAIT for the response to its query. The DEALLOCATE with *confirm* from **tp(b)** caused the response data to be sent from **tp(b)** along with a confirmation request message. **tp(a)** receives the response data with a status of data_complete. **tp(a)** does another RECEIVE_AND_WAIT and gets a status of confirm_deallocate indicating that **tp(b)** is waiting for a confirmation to the receipt of the response data in order to terminate the conversation. **tp(a)** uses the CONFIRMED verb to do this. **tp(b)** and **tp(b)** both terminate their ends of the conversation.

3.4.2 Conversation Verbs

The following list summarizes APPC conversation verbs by functional categories and indicates their use.

1. Start/Terminate

   ALLOCATE and DEALLOCATE respectively initiate and terminate conversations between procedures on logical units.

2. Send/Receive Data

   SEND_DATA and RECEIVE_AND_WAIT respectively send and receive data between procedures on logical units.

3. Change Link Direction

   REQUEST_TO_SEND and PREPARE_TO_RECEIVE allow a procedure to respectively ask for send permission or give up send permission.

4. Buffer Control

   FLUSH forces the contents of a buffer which holds conversation data to be sent on the network.

5. Conversation Synchronization

   CONFIRM requests a receiving procedure to send a confirmation that data was received. CONFIRMED allows the receiving procedure to send a confirmation.

6. Asynchronous Processing

   RECEIVE_IMMEDIATE, POST_ON_RECEIPT, TEST, and WAIT allow a procedure to interrogate the status of incoming data without being blocked. SEND_DATA never blocks.

7. Procedure Synchronization

   During a conversation, a procedure may invoke SYNCPT which sends a message to a receiving procedure to synchronize his activities on the receipt of the message. BACKOUT requests the receiving procedure to reset its state to the time of the last receipt of a SYNCPT message.

The Procedure Synchronization verbs are designed with database applications in mind. An example of their use is as follows. When a SYNCPT message is received, the receiving procedure applies all transactions received up to this point to its database. As new transactions arrive after receipt of the SYNCPT, they are applied to the database in such a manner that they can be *backed out*, i.e., the database can be restored to the condition it was in at the receipt of the last SYNCPT message. If something goes wrong during the conversation, the BACKOUT verb can be used by the procedures to reset their databases to a known point.

The application programming interface of APPC addresses the needs of two types of procedures which run on logical units, namely, network application programs and transaction programs. Network application programs are those written by application developers in high level languages. The verbs POST_ON_RECEIPT, TEST, WAIT, SYNCPT, and BACKOUT are not directly available for use in network application programs. Transaction programs implement system level functions on logical units and are intended for system programmers. All of the conversation verbs listed above are available for transaction programs. In addition, another class of verbs called the *control* verbs are available for use in transaction programs. Control verbs provide the functions necessary for the direct management of the communication hardware and other resources of the logical unit. Control verbs differ in their functionality according to a particular APPC implementation.

3.5 X Window System

The X Window System is a communication protocol and an application programming interface which defines an interface between user interactive devices (e.g., mouse, keyboard, display) and application programs. It was developed as part of Project Athena at MIT. The X Window System is a protocol in the Application and Presentation Layers and may be part of any protocol stack. However, most often, the X Window System is part of a TCP/IP protocol stack.

The X Window System enables a user at a workstation, the X server, to interact with several application programs, the X clients, which can be running on different nodes. For example, in figure 8, workstation **A** has two windows on its display. One window is controlled by client application $\mathbf{A_1}$ running on host $\mathbf{H_1}$. The other window is controlled by client application $\mathbf{A_2}$ running on host $\mathbf{H_2}$. Similarly, workstation **B** displays a window controlled by client application $\mathbf{A_1}$ on host $\mathbf{H_2}$ and another window controlled by client application $\mathbf{A_2}$ running on host $\mathbf{H_1}$. For both workstations **A** and **B** input to the appropriate application can be controlled by the location the mouse cursor on the display.

Note that, in the X Window System, the location of clients and servers is conceptually reversed from some of the other network services. In the example of figure 8, the workstation is the X *server* and the application programs residing on the hosts are the *clients*. From the point of view of other network services, such as, distributed file system or remote execution, the workstation is the client and the host is the server. In the example, the workstations could have attached part of the file systems of the hosts in which case, the workstations would be clients receiving file service from the hosts. Moreover, since the applications controlling the interactive devices of the workstations are running on remote hosts, the hosts are remote execution servers for the client workstations. Conceptually, what constitutes a *client* and what constitutes a *server* depends on the nature of the service relationship and not on whether a node is normally thought of as a *workstation* or a *host*.

The example of figure 8 also illustrates how the X Window System can be a participant in providing remote execution services in a heterogeneous computing environment. X client application programs can be implemented in any language on any systems as long as they adhere to the X protocol. Likewise, a workstation need only implement the X protocol by providing device handlers to its specific interactive devices. As a result, the workstation is able to run any available X client application on any node. In addition, remote execution services provided by means of Remote Procedure Call or Advanced Program to Program Communications could be used to invoke the remote X application and exchange data.

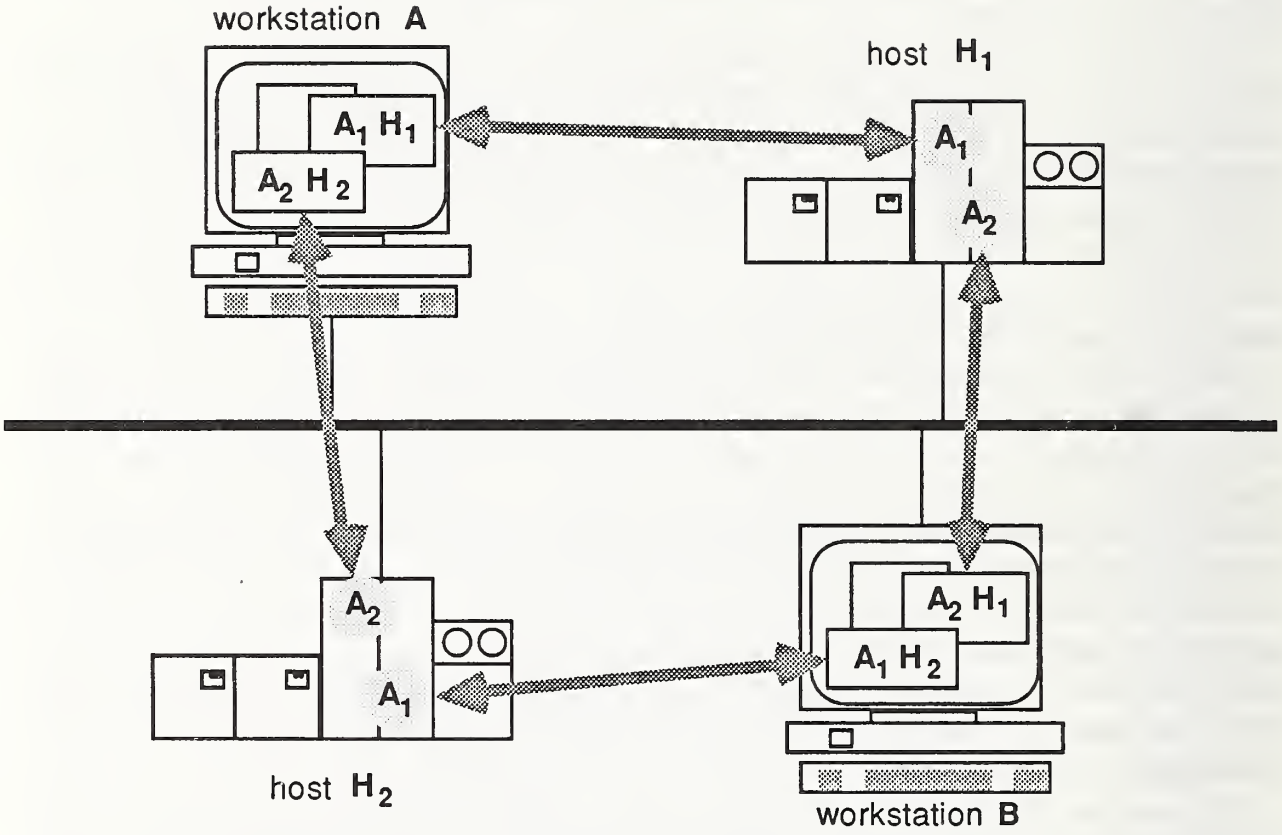


Figure 8. X Window System connectivity.

4 Conclusions

This report has noted the ever increasing demand on communication systems made by applications. From this demand has evolved the heterogeneous computing environment, a computer communications system made up of hardware and software components from many different producers.

It is useful to characterize heterogeneous computing environments from the point of view of the generic services which are provided to users who may be either application developers or end users. Heterogeneous computing environments provide services in three broad areas: distributed file system, distributed computing, and messaging. The distributed file system can apply to any type of device but most often supports mass storage devices and display devices. Distributed computing services include login, remote execution, transaction processing, and cooperative processing. Messaging services include mail and conferencing.

Heterogeneous computing environments can only exist because of the communication protocol, application programming interface, and user interface standards agreed upon by users and producers. Some examples of such standards have been presented. Standards which enable distributed file systems include Network File System and External Data Representation. Standards which permit distributed computing include Remote Procedure Call, Advanced Program to Program Communication, and the X Window System. In addition to supporting distributed computing services, the X Window System also supports the implementation of user interfaces.

A References and Related Reading

Advanced Program to Program Communication for the IBM Personal Computer Programming Guide, First Edition, Part Number 61X3813, IBM Corporation, February 1986.

Ahuja, V., "Common Communications Support in Systems Application Architecture", *IBM Systems Journal*, **27**, 3, 1988.

Balkivich, E., Lerman, S., Parmelee, R. P., "Computing in Higher Education: The Athena Experience", *Communications of the ACM*, November 1985.

Birrell, A. D., Nelson B. J., "Implementing Remote Procedure Calls", *ACM Transactions on Computer Systems*, February 1984.

Cheriton, D., "The V Distributed System", *Communications of the ACM*, March 1988.

Demers, R. A., "Distributed Files for SAA", *IBM Systems Journal*, **27**, 3, 1988.

Gifford, D. K., Needham, R. M., Schroeder, M. D., "The Cedar File System", *Communications of the ACM*, March 1988.

Government Open Systems Interconnection Profile (GOSIP), Federal Information Processing Standards Publication 146, National Institute of Standards and Technology, 1989.

Grey, J. P., Hansen, P. J., Homan, P., Lerner, M. A., Pozefsky, M., "Advanced Program to Program Communication in SNA", *IBM System Journal*, **22**, 4, 1983.

Hedrick, C. L., "Introduction to the Internet Protocols", Laboratory for Computer Science Research, Computer Science Facilities Group, Rutgers University, July 3, 1987.

IEEE Standard Portable Operating System Interface for Computer Environments (POSIX), IEEE Std 1003.1-1988, Institute of Electrical and Electronics Engineers Inc., 1988.

International Standard ISO 7498 – Information Processing Systems – Open Systems Interconnection – Basic Reference Model, Reference Number:ISO 7498:1984(E), ISO TC97/SC21 Secretariat, ANSI, New York, New York, 1984.

International Standard ISO 8571-1 – Information Processing Systems – Open Systems Interconnection – File Transfer, Access and Management, Reference Number:ISO 8571-1:1987(E), ISO TC97/SC21 Secretariat, ANSI, New York, New York, 1987.

International Standard ISO 8824 – Information Processing Systems – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1), Reference Number:ISO 8824:1987(E), ISO TC97/SC21 Secretariat, ANSI, New York, New York, 1987.

International Standard ISO 8825 – Information Processing Systems – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1), Reference Number:ISO 8825:1987(E), ISO TC97/SC21 Secretariat, ANSI, New York, New York, 1987.

Liskov, B., "Distributed Programming in Argus", *Communications of the ACM*, March 1988.

McWilliams, G., "Developers Ponder Choices Among Graphic Environments", *DATAMATION*, August 15, 1988.

Morris, J. H., Satyanarayanan, M., Conner, M. H., Howard, J. H., Rosenthal, D. S. H., Smith, F. D., "Andrew: A Distributed Personal Computing Environment", *Communications of the ACM*, March 1986.

Network Programming, Sun Microsystems, Part Number:800-1779-10, Revision A, May 9, 1988.

"Next-generation NOSs Already Are Yielding Practical Benefits", *Data Communications*, May 1988.

Notkin, D., Black, A. P., Lazowska, E. D., Levy, H. M., Sanislo, J., Zahorjan, J., "Interconnecting Heterogeneous Computer Systems", *Communications of the ACM*, March 1988.

Notkin, D., Hutchinson, N., Sanislo, J., Schwartz, M., "Heterogeneous Computing Environment: Report on the ACM SIGOPS Workshop on Accommodating Heterogeneity", *Communications of the ACM*, February 1987.

POSIX: Portable Operating System Interface for Computer Environments, Federal Information Processing Standards Publication 151-1, National Institute of Standards and Technology, 1989.

Quarterman, J. S., Hoskins, J. C., "Notable Computer Networks", *Communications of the ACM*, October 1986.

Reinsch, R., "Distributed Database for SAA", *IBM Systems Journal*, **27**, 3, 1988.

Rifkin, A. P., Forbes, M. P., Hamilton, R. L., Sabrio, M., Shah, S., Yueh, K., "RFS Architectural Overview", *Summer Usenix Conference Proceedings*, Atlanta 1986.

Roux, E., "OSI's Final Frontier: The Application Layer", *Data Communications*, January 1988.

"SAA: Big Blue's Distributed Processing Road Map Takes Place", *Data Communications*, November 1987.

Satyanarayanan, M., "A Survey of Distributed File Systems", *Annual Review of Computer Science*, Volume 4, 1989.

Scheifler, R. W., Gettys, J., "The X Window System", *ACM Transactions on Graphics*, April 1986.

Scherr, A. L., "SAA Distributed Processing", *IBM Systems Journal*, **27**, 3, 1988.

Spanier, S., "Comparing Distributed File Systems", *Data Communications*, December 1987.

Stevens, E. E., Bernstein, B., "APPC: The Future of Microcomputer Communications within IBM's SNA", *Data Communications*, July 1986.

Svobodova, L., "File Servers for Network-based Distributed Systems", *ACM Computing Surveys*, December 1984.

System V Interface Definition, Volumes 1–3, AT&T, 1986.

Tannenbaum, A. S., **Computer Networks**, Prentice-Hall Inc., 1981.

☆U.S.GOVERNMENT PRINTING OFFICE:1989-261-913/00102

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NIST/SP-500/176	2. Performing Organ. Report No.	3. Publication Date November 1989

4. TITLE AND SUBTITLE

Introduction to Heterogeneous Computing Environments

5. AUTHOR(S)

John F. Barkley, Karen Olsen

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
(formerly NATIONAL BUREAU OF STANDARDS)
U.S. DEPARTMENT OF COMMERCE
GAITHERSBURG, MD 20899

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

National Institute of Standards and Technology
Gaithersburg, MD 20899

10. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number: 89-600784

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

Computer networks are becoming larger not only in the number of nodes connected but also in the geographic area spanned. In addition, networks are becoming more diverse in the variety of equipment from which the network is implemented. This report provides an introduction to the concept of a *heterogeneous computing environment.* It characterizes heterogeneous computing environments from the point of view of the generic services provided. Standards are necessary in order to implement heterogeneous computing environments. The report provides an introduction by example to the types of technical standards that are necessary in a heterogeneous computing environment and illustrates how such standards can be used to provide services.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

Communications; computer; distributed computing; networks; protocols

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☒ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☐ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

37

15. Price

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SYSTEMS TECHNOLOGY

Superintendent of Documents,
Government Printing Office,
Washington, DC 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Institute of Standards and Technology Special Publication 500-.

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

(Notification key N-503)

NIST Technical Publications

Periodical

Journal of Research of the National Institute of Standards and Technology—Reports NIST research and development in those disciplines of the physical and engineering sciences in which the Institute is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Institute's technical and scientific programs. Issued six times a year.

Nonperiodicals

Monographs—Major contributions to the technical literature on various subjects related to the Institute's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NIST, NIST annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NIST under the authority of the National Standard Data Act (Public Law 90-396). NOTE: The Journal of Physical and Chemical Reference Data (JPCRD) is published quarterly for NIST by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements are available from ACS, 1155 Sixteenth St., NW., Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Institute on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NIST under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NIST administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NIST research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

Order the **above** *NIST publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*

Order the **following** *NIST publications—FIPS and NISTIRs—from the National Technical Information Service, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NIST pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NIST Interagency Reports (NISTIR)—A special series of interim or final reports on work performed by NIST for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Service, Springfield, VA 22161, in paper copy or microfiche form.

U.S. Department of Commerce
National Institute of Standards and Technology
(formerly National Bureau of Standards)
Gaithersburg, MD 20899