Investigating Integration in Distributed Systems

by

Scott R. Fraser

A thesis
presented to the University of Manitoba
in partial fulfillment of the
requirements for the degree of
Master of Science
in
Computer Science

Winnipeg, Manitoba

INVESTIGATING INTEGRATION IN DISTRIBUTED SYSTEMS

BY

SCOTT R. FRASER

A thesis submitted to the Faculty of Graduate Studies of
the University of Manitoba in partial fulfillment of the requirements
of the degree of

MASTER OF SCIENCE

I hereby declare that I am the sole author of this thesis.

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Scott R. Fraser

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Scott R. Fraser

ABSTRACT

Distributed systems have been the subject of much research. Integration, the most important unifying concept of all, allowing a high degree of interaction and cross fertilization among distributed system components, and enabling the different areas and technologies to mesh together, has been given less emphasis. This thesis undertakes to investigate integrated distributed systems and the application level protocols, specifications, and service elements which will sufficiently integrate their components in order to reduce user interface complexity, control the flow of information, and enhance the overall efficiency of the environment.

The issues to be resolved in integrating a distributed system include naming, sharing, protection, synchronization, intercommunication, and error recovery. These and other issues are applicable to the application layer of the Open Systems Interconnection Reference Model because they are the usual services needed in various forms by all applications. We examine each of these issues and use them to describe a hypothetical integrated distributed system.

ABSTRACT

Distributed systems have been the subject of much research. Integration, the most important unifying concept of all, allowing a high degree of interaction and cross fertilization among distributed system components, and enabling the different areas and technologies to mesh together, has been given less emphasis. This thesis undertakes to investigate integrated distributed systems and the application level protocols, specifications, and service elements which will sufficiently integrate their components in order to reduce user interface complexity, control the flow of information, and enhance the overall efficiency of the environment.

The issues to be resolved in integrating a distributed system include naming, sharing, protection, synchronization, intercommunication, and error recovery. These and other issues are applicable to the application layer of the Open Systems Interconnection Reference Model because they are the usual services needed in various forms by all applications. We examine each of these issues and use them to describe a hypothetical integrated distributed system.

ACKNOWLEDGEMENTS

My thanks to Dr. Matthias Laucht, who motivated and supervised the research for this thesis.

A special thanks to Bob Morin for his continued enthusiasm in my work, and for his many welcomed contributions toward the content of this thesis.

Finally, I gratefully acknowledge the support given to this work by the University of Manitoba and the National Sciences and Engineering Research Council of Canada.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter I

INTRODUCTION

Distributed systems have been the subject of much research. Integration, the most important unifying concept of all, allowing a high degree of interaction and cross fertilization among distributed system components, and enabling the different areas and technologies to mesh together, has been given less emphasis. This thesis undertakes to investigate integrated distributed systems and the application level protocols, specifications, and service elements which will sufficiently integrate their components in order to reduce user interface complexity, control the flow of information, and enhance the overall efficiency of the environment.

In the early computer systems operation was strictly in batch mode. The data or programs were prepared off-line (that is, on simple mechanical devices such as card or tape punches) and submitted to be run at the computer operator's discretion. When operating systems were introduced, these replaced the work of human operators in scheduling work so that it was done in an efficient and cost-effective manner. Because of the high cost of early computers and the fairly limited use made of them, one computer would be required to serve many users. A single central site might run the computer, with users located both at the centre and dispersed at other subsidiary sites. This approach to computing can still be seen in operation. Its faults were the

remoteness of the system from the user and the concentration of system design into one unit. The remoteness, for example, meant that dealing with errors and exceptions accounted for a large part of the system. The concentration meant that all the parts of the system could interact without an overall discipline and the system could easily become untidy and unmanageable.

Remoteness was the first problem to be overcome and the first steps, leading to the first computer network, seemed easy. Some computer terminals were teleprinters and these could be moved to distant locations and connected, by cables or modems and telephone lines. This began to happen as soon as multi-access or "time-sharing" - the dividing of the power of one processor between many almost independent users - had been achieved. It formed the earliest kind of network and gave remote access to a central computer.

Such a network is centered around one computer and in the early networks the computer provided a single function, such as banking or airline reservations. The centralized systems were found to be inflexible when their owners tried to add new functions or operate more than one central computer to provide the services.

The old model of a single computer serving all of an organization's computational needs was rapidly being replaced by one in which a large number of separate but interconnected computers did the job. These systems are called computer networks.

Modern computer networks tend to be heterogeneous, using computers and terminals from many manufacturers. Unfortunately, no single

standard governs their interfaces, formats and procedures. By itself, this would prevent any useful interaction in a network, so a large part of a network design is concerned with formalizing the ways in which its various active components will interact.

One of the first attempts at solving the network design problem was through Open System Interconnection (OSI) - a concept by which any system can be interconnected with any other system, even if they are made by different manufacturers. In order to do this, both systems must be open, that is, both must use a standard set of rules or protocols to communicate. The OSI concept was developed because the promise of networking was hampered by the proliferation of standard and non-standard protocols that mixed different sets of functions, or did the same functions in different ways. The International Standard's Organization attempted to define an overall architecture for all levels of protocol in a network. Thus, instead of concentrating all system design into one unit, the architecture was divided into seven layers: the four lowest concerned with basic transport services and the three highest dealing with network applications and the users.

This reference model represents a middle-of-the-road approach to computer network design, which aims to take full account of heterogeneity in the real world, while making interconnection more or less straightforward.

The overall goal of network architecture design is to meld the diversity of resources and protocols into some coherent form so that the collective human/software/hardware resources can intercommunicate. The

common current approach is to create a transport or end-to-end protocol built on lower-level protocols that move uninterpreted blocks of data, and then construct, on top of this service, protocols for whole-file transfer, virtual terminal access, and other virtual device usage.

However, the full potential of computer networking for resource sharing and distributed computing cannot be realized with such an architecture for the following reasons. First, each programmer wishing to provide or use a new network-sharable resource must face anew all the issues of data-type translation, naming, protection, and interfacing to the transport layer. Second, a terminal user must know the various naming and other access mechanisms required by the network, each host, and each service. He must login to his local host, use a network access program, and then login to his target host(s), each probably using different conventions. And third, the setting up of accounts and other administrative procedures are awkward. The user must explicitly establish accounts and receive billing from each administration controlling a host with resources he wishes to use.

The next step for the elimination or minimization of these and other problems is the creation of a distributed system. A distributed system is an improvement to the basic design of a computer network, allowing for a high degree of cohesion, transparency, and decentralization.

A common view of distributed system design is that it attempts to create the illusion of a nondistributed system at some level. In other words, the user of a distributed system should not be aware that there are multiple processors, or that his computations and data are dispersed among various physical locations.

At this point, the reader should be made aware of two trends that were particularly evident as we progressed from centralized computer systems to distributed systems. First, the complexity and sophistication of hardware, software, procedures, and protocols have increased. This is because we are trying to unify a number of separate heterogeneous computers instead of simply interconnecting them in some ad hoc fashion. At the same time, the proliferation of computer systems of both small, medium, and large scale, and the dramatic change in the relative cost of computing versus communications, has given the user a myriad of aids for information storage, retrieval, communication, manipulation, and control. This too increases the overall complexity of the system. Second, the level of integration has increased. However, this integration has not progressed at the same rate as the conglomeration of automated aids and applications sweeping the distributed environment. Integration, the most important unifying concept of all, allowing a high degree of interaction and cross fertilization, and enabling the different areas and technologies to mesh together, has been given less emphasis.

Integration is formally defined as the combination of parts to create a whole [Sykes76]. This implies some form of synergy in which the system components are tied together into a whole that is greater than the sum of its parts. The value added is directly proportional to the degree to which the different functions of these components are integrated into a total system.

If we extrapolate and improve upon the potentials of a distributed system, we get an integrated distributed system. Integration here takes

place on at least two fronts: functional integration and system integration. Functional integration refers to the need for the user's model of a system to be complete and consistent. System integration refers to the need for the operating systems, programming languages, architecture, and data bases that converge into a single, uniform environment.

The issues to be resolved in integrating a distributed system include naming, sharing, protection, synchronization, intercommunication, and error recovery. These and other issues are applicable to the application layer of the Open Systems Interconnection Reference Model because they are the usual services needed in various forms by all applications. We examine each of these issues and use them to describe a hypothetical integrated distributed system.

1.1 ORGANIZATION OF THESIS

Chapter 1 serves as a quick introduction to the thesis.

Chapter 2 familiarizes the reader with network classification schemes, architectures, and protocols. The International Standards Organization's reference model for Open Systems Interconnection is introduced as a basis for taking a layered approach to the development of network and distributed system architectures. A great deal of emphasis is placed on the application layer, both in this chapter and succeeding chapters because it is only through the services provided at this layer that integration and its full potentials can be realized. Two popular network architectures, Systems Network Architecture and DECNET, are also briefly discussed in this chapter.

Chapter 3 introduces the concept of distributed systems, and deals with those application layer issues most appropriate for the development of integrated distributed systems. The chapter outlines the nature of the issues, and touchs on some of the difficulties faced by distributed systems.

Chapter 4 revisits the application layer and introduces a major recurring theme in this thesis: application layer service elements. Common application layer service elements provide capabilities needed by the application processes for proper system-system interaction, independently of the nature of the applications. System service elements provide capabilities to handle the needs of specific information transfer processes. User specific service elements provide capabilities to specific application processes. All three are identified and the most important elements in each category are briefly outlined.

Chapter 5 introduces the concept of an integrated distributed system. A summary of some of the more important issues discussed throughout the thesis and their applicability to integrated distributed systems is presented. The issues are subsequently used to describe a hypothetical integrated distributed system.

We conclude in Chapter 6, summarizing what we have covered, what we think we know about distributed systems and integration, and what we feel is necessary to bring integration closer to reality.

Chapter II

COMPUTER NETWORKS

Finding a suitable definition for a computer network can be a difficult task. Fortunately, both Tanenbaum and Pooch have provided us with the elements necessary for a simple and yet complete definition.

> **A Computer Network is an interconnected collection of autonomous computers.**

By interconnected we mean the computers can exchange meaningful information, and by autonomous we mean that each computer in the network is its own master, with the ability to accept or reject a message based on its own knowledge of its own status. For example, if one computer system can forcibly control another one, the computers are not autonomous.

A computer network consists of the following components:

- Hosts,
- Terminals,
- Nodes,
- Transmission links.

Hosts are the autonomous computers mentioned above, and are usually intended for running user application programs. Large multiprogrammed mainframes or small microcomputers are two examples of hosts.

Terminals are devices interfacing the user to the computer or computer network.

Nodes (or switches) are specialized computers whose primary function is to switch data. This means the node is capable of receiving data and performing some operation on them. An operation may be to determine the optimal route or path from the current node to the desired destination of the data (which may be another node or a host).

Transmission links, also called lines, channels, or circuits, permit the flow of information between the nodes of the network (for example, copper wire, lasers, fiber optics, microwave, satellite, and so on).

Together, the nodes and links as well as control software, make up the communication subnet (often called the data network, communication network, or simply the subnet). By separating communication activity (the subnet) from user application activity (the hosts), computer network design and complexity is reduced considerably (refer to Figure 2.1).

From this point on, when we refer to a computer network, we are encompassing both the communication subnet as well as the co-operating hosts connected to it. This sets the terminology for the remainder of the thesis.

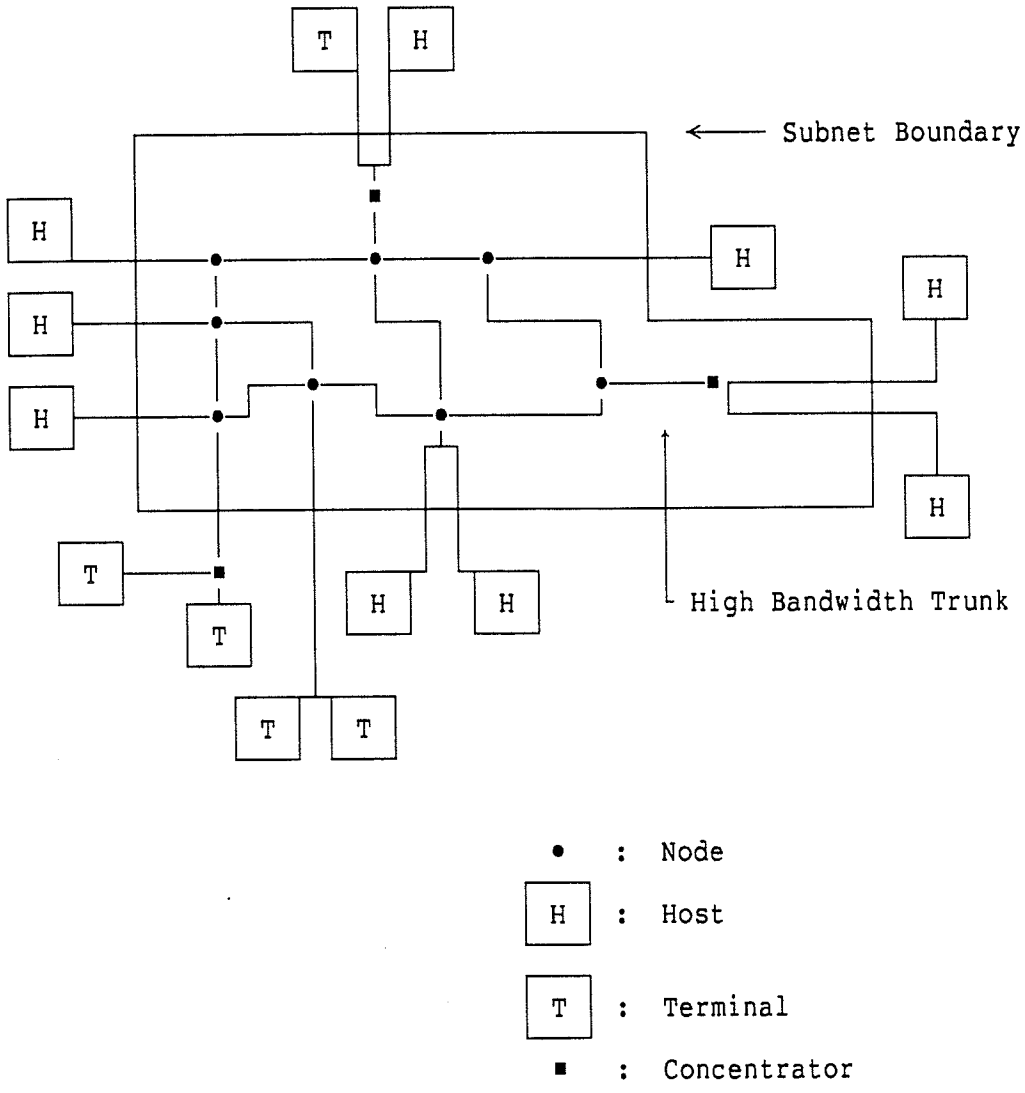


Figure 2.1: A Typical Point-to-Point Long Haul Network

2.1 NETWORK CLASSIFICATION SCHEMES

A computer network can be classified according to:

- topology of the subnet (that is, node arrangement),
- composition (that is, homogeneous or heterogeneous),
- data communication (switching) discipline of the subnet,

- distance between the hosts (geography),
- message routing procedures of the subnet,
- major functions the network performs,
- ownership,
- any combination of the above.

In this section we would like to discuss a number of these classification schemes in more detail.

2.1.1 Topology

Network topology is a means of categorizing data communication networks (that is, the subnets). Topology refers to properties of a subnet that are independent of its shape and size, properties such as the connection pattern and arrangement of the links and nodes.

Two basic connection schemes exist in the subnet: point-to-point and multipoint connections. With a point-to-point connection, there exists a unique connection between individual pairs of nodes within the subnet (see Figure 2.2(a)). However, to move information between two points which are not directly connected, a link must be established. This link may contain several connecting point-to-point paths, and possibly an intervening device. Figure 2.2(b) illustrates the concept of a multipoint connection. In this configuration, a node may simultaneously connect to more than one node. It is possible to "broadcast" information to a number of nodes simultaneously. In a point-to-point system, a similar logical function of broadcasting is possible, but the "broadcast" must occur sequentially.

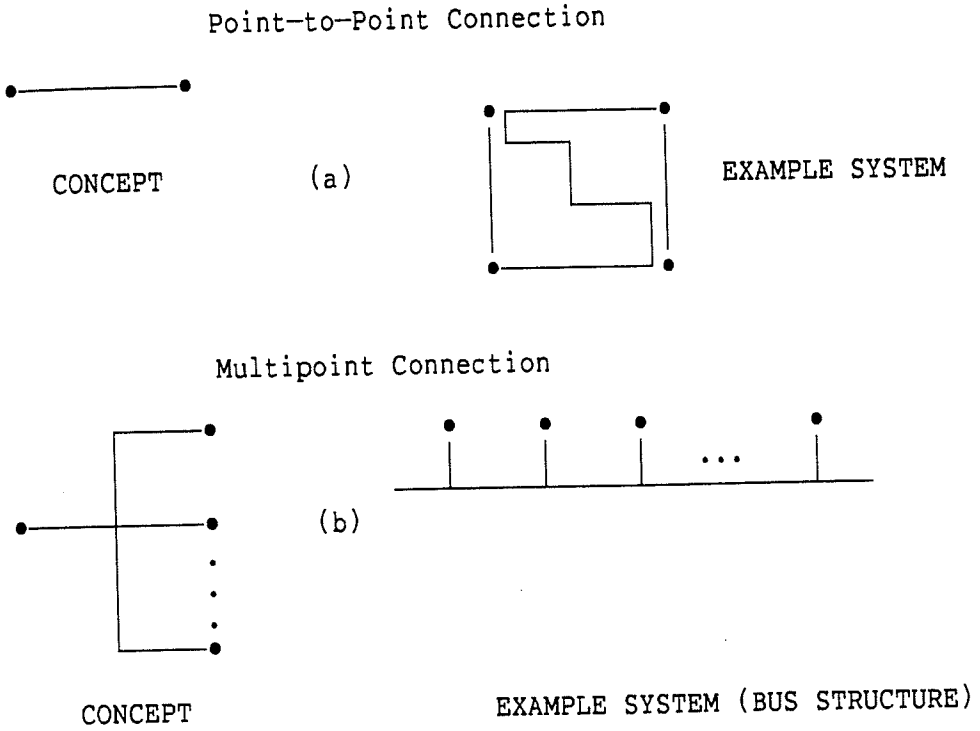


Figure 2.2: Point-to-Point and Multipoint Concepts

2.1.2 Composition

The composition of a network is yet another important classification scheme, with networks being either homogeneous or heterogeneous.

By homogeneous, we mean that each of the computers in a network is similar. Homogeneity usually implies computers of similar architecture, usually from one manufacturer and sometimes even from one product line. A homogeneous computer network has similar hosts and a homogeneous communication subnet has similar nodes. An SNA network is an example of a homogeneous computer network with a homogeneous subnet.

By heterogeneous, we mean that the interconnected computers are dissimilar, that is, usually of different architectures, from different

manufacturers. Because nodes in a communication network should interact with each other in an integrated, efficient manner, it is less common for subnets to be heterogeneous than it is for computer networks. For example, the CYBERNET network is heterogeneous and its subnet is homogeneous.

Heterogeneity is desirable in general because the user has available a rich diversity of computing power. However, as the reader shall see shortly, heterogeneity of components introduces new problems to the system. Translation issues resulting from heterogeneity of file names, access procedures, and data representations is an example of one such problem.

2.1.3 Data Communication Discipline

In this section we discuss the classification of communication subnets in terms of the discipline used for the physical transmission of data between nodes.

POINT TO POINT

Point-to-point communication subnets can be further classified as being either circuit, message, or packet switched. Let us discuss each of these in turn.

Circuit Switching

With circuit switching, a complete path through the network with a fixed amount of transmission capacity (bandwidth) is reserved when the sender initiates a communication, and is released only when communication terminates. A good example of this is the telephone (voice) system. If a user dials a time sharing facility, a complete dedicated circuit is established after dialing and remains intact until the user or the time sharing facility hangs up. Even if communication between source and destination is idle, charges will continue to accumulate.

As a general rule, circuit switching is best suited for long messages passing at a steady rate. On this basis, circuit switching would be the choice for digitized voice, video data, and bulk data transfer, and would be less appropriate for applications requiring relatively short setup times (such as point-of-sale credit verification).

TYMNET was the first commercial distributed circuit switching network and spanned the United States and Europe connecting 57 cities using 80 nodes and 37 large-scale hosts as of 1973 [Pooch83].

Increased switching speeds and more sophisticated carrier services may see a change to the viability of circuit switched subnet designs in the future. For example, the old telephone exchanges use electromechanical contacts and can therefore take seconds to set up the switched path. Modern circuit switching equipment uses solid-state electronics with no moving parts and can set up paths in milliseconds (this is sometimes called fast-connect switching). In this case, the

circuit is set up quickly, is usually sustained only for the time it takes one message to pass, and is released very quickly.

Message Switching

A message is a logical unit of information (such as a bit string) for the purposes of communicating to one or more destinations [Pooch83]. A message consists of a header field containing control information necessary for maintaining the control of the network operations involved in the delivery of the message; the information field or body containing the actual data to be transmitted; and the trailer field delimiting the end of the message. A telegram, program, or data file are all examples of a message.

A message switched subnet is one in which each message is passed between nodes on a message-by-message basis. This means that circuits are no longer dedicated for exclusive use by one sender/receiver pair as with circuit switching. A message, which may be broken down into blocks of data, must be either received at a node in its entirety, or cancelled across a link before the next message can be sent. Each block of a message must be sent to the receiving node in sequence, so the node can rebuild the message, check it for errors, and verify to the sending node that it has been received properly. If the message is not received correctly, it must be retransmitted until the verification is successful - only then can the message be sent to the next node. As well, only the first block of a message has control information in it for further routing.

For the regular user, message switching is less expensive than circuit switching because circuit costs are divided among the users sharing the system. However, message switching can cause unacceptable delay for the real-time user, justifying the need for a more sophisticated switching technique.

CYBERNET, a public computer communication network designed and implemented by Control Data Corporation is an example of a message switching network [Pooch83].

Packet Switching

Packet switching has evolved over only the past ten years. As with message switching, each message is divided into blocks (called packets) and transmitted between subnet nodes. However, with packet switching, and unlike message switching, each packet is sent as an independent entity; each contains its own control information to direct it through the network, independently of all other packets belonging to the same message.

Resource requirements for each node are greatly reduced because a packet can be transmitted immediately at each node without having to wait for the entire message. As well, by analyzing system data load, the subnet can dynamically alter routes to minimize system load.

Packet switching best suits short messages sent at irregular rates and therefore is a good choice for terminal traffic (typical communication between humans and computers in an interactive real-time

environment is bursty in nature). It is less suitable for bulk data transfer or for real time voice or video data.

Some well known packet switched networks include the Advanced Research Projects Agency (ARPA) network called ARPANET, TELENET, and DATAPAC.

MULTIPOINT, BROADCAST

The basic concept behind broadcast or multipoint transmission is that the source transmits to more than one receiver.

Although the circuit and packet switching techniques were discussed in a point-to-point environment, they may also apply to broadcast channels. For example, the broadcast network is circuit switched if a portion of the channel is dedicated to a conversation throughout its duration, without regard to actual usage. If the channel is dynamically requested, used, and released for every packet, then the network is packet switched.

Broadband (BRB) and baseband (BAB) are the most common of the multipoint technologies, therefore, we would like discuss each of them at this time.

Broadband technologies use community antenna television, or cable television (CATV) technology because it is cheap, provides a reliable communication path, and is easy to split into channels. All signal information in the system is converted to radio frequency (analog) form, making it easy to subdivide the signals into subfrequencies. For

example, in Canada and the United States, the basic television channel is 6 MHz, therefore, with a 300 MHz cable, 50 channels are possible. Channels need not be the same size. Because of this, broadband systems may include voice, data, and video signals. Wang offers the Wangnet broadband local area network. It uses a coaxial carrier with a data rate of 340 MHz, but the range 0 to 10 MHz is excluded due to noise problems [Chora84].

With baseband technology, one signal, usually digital, is sent through the system, therefore, unlike broadband, the entire bandwidth is used. Baseband data rates usually are between 1 and 10 Mbps. Since the signal is digital, baseband technology is used primarily for data transmission. Two systems using baseband transmission are Datapoint's Attached Resource Computer (ARC) communication facility using coaxial cable links with a data rate of 2.5 Mbps, and Xerox's Ethernet, using coaxial cable with a data rate of 10 Mbps.

2.1.4 Geography

Communication subnets can be classified according to the distance separating the nodes in the communication system (and hence the hosts in a computer network using that subnet). Two common geographical classifications are local area networks (LANs) and long haul networks (LHNs).

In a LAN environment, the hosts are contained in a single building or complex of buildings connected by high-bandwidth cable or other communication media designed for this purpose. The system usually spans

between 10 meters and 10 kilometers, and because of this limited distance, it is nearly always owned by a single (private) organization.

A popular LAN implementation is Xerox's Ethernet, a broadcast (baseband), packet switched LAN using bus topology and coaxial cable for packet transmission. Ethernet is a communication subnet, and with the addition of hosts, can be used to form computer networks.

LHNs connect nodes in different cities using media such as twisted pair, coaxial cable, optical fiber, terrestrial microwave links, earth satellites, and so on. Node separation ranges from a few kilometers to around the globe. As well, LHNs usually involve two or more organizations: the carrier operating the communication subnet, and the owner(s) of the hosts (the users).

Nodes connected by telephone lines may be right on the carrier's premises, each node servicing two or more hosts. To save on long distance leased-line charges, host and terminal communication may be funnelled through a remote concentrater (as shown in Figure 2.1). Nodes may also be connected by satellite, in which case the nodes may be located on the customer's premises.

ARPANET, TELENET, DATAPAC, and the public telephone system are all examples of long haul computer networks.

2.1.5 Other Network Classification Schemes

In addition to the aforementioned classifications, a computer network may be categorized according to message routing procedures, function, or ownership.

Routing procedures are concerned with which output line an incoming packet in the subnet should be transmitted on. For example, deterministic routing algorithms derive routes according to some prespecified rule that is generally designed to optimize a particular topological configuration. On the other hand, stochastic routing algorithms operate as probabilistic decision rules as opposed to deterministic rules.

Some networks are designed for a specific function or application, like electronic funds transfer or airline reservations. By taking on a specific requirement rather than a general-purpose one, simpler and cheaper control mechanisms are needed for the network.

Networks can also be classified according to whether they are private or public. Private networks are usually built within one corporation or government organization, and the communication facility is intended for use only by that organization. LANs are typically private subnets. Public networks on the other hand, are built by the common carriers, and the transmission and switching facilities are shared by the hosts of many corporations.

2.2 NETWORK ARCHITECTURES AND PROTOCOLS

2.2.1 Architecture Versus Implementation

A conceptual framework is needed for organizing the discussion and specification of computer network design goals, issues, and interrelationships, and for providing some common terminology to be used in the following chapters. We refer to this framework as the network architecture and distinguish it from network implementation.

The precise definition of the functions that a computer network and its components should perform is its architecture [Green80]. Specifying a network architecture requires decomposing the entire system and specifying the services to be offered by components of the structure and the relationship between these components. Today, a common way of looking at networks is in terms of layered architectures. Such a network architecture consists of a hierarchy of independent layers that contain modules for performing defined functions. We will elaborate on this important concept shortly.

The precise definition of the hardware and software components needed to implement the architecture is an activity associated with network implementation. Network implementation uses the architecture as the rulebook upon which to base its functionality.

Even though the implementation should adhere to the architecture, their structures may be quite different. For example, layering is commonly carried out through the architecture design stages, but may not necessarily be reflected in the implementation. For performance reasons, layering at this stage may be inappropriate, causing undue overhead in and across the layers.

Now let us focus our attention on the framework and terminology associated with a layered architecture. We will deal with architecture considerations only, leaving implementation considerations as an exercise for the design engineer.

2.2.2 The Technique of Layering

The idea that a system could be built up in layers or subdivided into layers is not new. Many individual systems have evolved with an implicit layered structure. The evolution of a computer system from the basic hardware is a typical example.

Early systems began with the central processor and any functions needed by the user had to be coded by the user. In time, an executive or nucleus layer was added to allow the user to communicate with the central processing unit (CPU) and the peripherals. Operating system (OS) software was then developed to handle system management functions, eventually allowing users to share the machine's resources simultaneously. Soon there developed a need for system service programs; utilities such as data base management systems (DBMSs), assemblers, compilers, text editors, which all ran under the supervision of the operating system. The application program then became the highest level of this hierarchy, operating either subordinate to one of the service programs, or to the operating system. Each of the layers is distinct from the others in the way it operates and in the way it was developed. Yet each succeeding layer is dependent on the services provided by supporting layers for proper operation. For example, a data base management system needs the services of the operating system to access data from disk. Each layer has been separately developed and could be removed or replaced by a functionally equivalent layer which perhaps operates internally in a completely different manner.

This natural layering structure increases the usefulness and ease of development of the system, by dividing complex functions into simpler, more manageable modules.

This same concept can be seen in modern computer networks. Complexities associated with network architecture and implementation, can be managed more effectively, by organizing the components of the network into layers or levels. The diverse functions offered by the network are split into manageable pieces. Each successive layer offers services to the higher layers, and each layer remains a "black box", hiding its specific implementation details. Another major advantage of this organization is that the complexity can be hidden from the network users.

One disadvantage of layering is that it may create more work for the network designer. Certain functions may have to be duplicated over a number of layers (such as flow control) and this in turn adds to the amount and complexity of software and/or hardware being developed.

2.2.3 Protocols, Interfaces and Services

The concepts of protocol, interface, and service are fundamental to the understanding of how two layered systems communicate with each other.

When two systems want to exchange data directly, certain rules must be developed to ensure an orderly and relatively error-free conversation. In fact, the two systems must come to prior agreement about these rules. For example, these rules may include information

about who will initiate and who will terminate the conversation, and whether there are to be data with special meaning, and if so, how are the data to be handled. The set of rules and conventions governing exchange of data between systems is known as a protocol.

Logically, each system interacts with the other using the protocol, and the protocol exists only between the peer (corresponding) layers on the different systems. The set of rules governing the exchange of information between adjacent layers is called an interface.

Finally, except for the highest layer in a system which operates for its own purpose, each layer contains a service for the provision of certain functions or facilities to the layer above. In other words, the service from any layer adds value to the service provided by the adjacent lower layer and offers this value-added service to the layer above.

Together, the layers, interfaces, and protocols for a computer network form a network architecture. The details of implementation are not part of the architecture.

The process by which two systems exchange data via some media can be described as follows. In the general case of a system constructed of layers, there is no direct physical horizontal data transfer from a layer on one system to the corresponding layer on the other system, except for the lowest. Instead, data and control information are transferred vertically down across the boundary between adjacent layers, until the lowest layer is reached. Below this layer exist the actual physical media by which data are sent to the receiving machine.

A simple example might clarify the concepts most important to a multilayered architecture.

Two young entrepreneurs live two miles apart each in his own three story penthouse apartment. Let us refer to the business partners as Mr. A and Mr. B. Both have been tardy with their telephone payments, and so they have no telephone connection with the outside world. Mr. A wants to tell his partner that their business is on the verge of bankruptcy. Mr. A's secretary just quit on him, so he must pen the message himself, but he can only write in Gregg shorthand. He completes the correspondence and sends it down on the elevator to the second floor. At this point the maid translates his message into English and sends it down on the elevator to the chauffeur waiting on the main floor. Using Mr. A's Rolls-Royce Corniche Convertible, the chauffeur drives to Mr. B's apartment, where the message is handed to the bodyguard on the main floor. Mr. B's bodyguard sends the note up on the elevator to the butler on the second floor. Since Mr. B can only read in Pitman shorthand, the butler first converts the message appropriately before sending it up to the third floor. Mr. B reads the message and faints.

This scenario illustrates certain key points. First, it shows the existence of a physical interface between adjacent floors in the same apartment (that is, the elevators).

Second, it shows the existence of a logical protocol between peer floors. For example, the maid and the butler logically communicate using a defined protocol. The maid could have translated the message to French or German, as long as the butler had agreed with such rules.

Third, the scenario shows the existence of the low level physical medium at the main floor between apartments. It is important to realize that Mr. A and Mr. B are concerned only with the fact that a connection exists. They are not concerned with how the data are exchanged between them. The chauffeur could have accomplished the same task by riding a bicycle between apartments. In general, the caller is unaware and unconcerned with the means by which the data exchange is accomplished, as long as what he submits to the sending system is delivered intact to the receiver.

2.3 THE ISO MODEL OF ARCHITECTURE FOR OPEN SYSTEMS INTERCONNECTION

Experimental networks like ARPANET were conceived as heterogeneous from the start. Each manufacturer developed his own conventions, or network architecture, for interconnecting his own equipment. In 1978, the International Standards Organization (ISO) recognized the need for interconnecting systems from different manufacturers and formed a new subcommittee (SC16) for Open Systems Interconnection (OSI) to help develop an architecture which could serve as a framework for the definition of standard protocols. If a system conforms to these standards, it will be "open" to all other systems obeying the same standards throughout the world [Zimme78].

After some discussion, SC16 agreed on a layered architecture comprising seven layers (physical, data link, network, transport, session, presentation, and application). In July 1979, specifications of this architecture and recommendations to develop an initial set of standard protocols for OSI, were passed under the name of "OSI Reference

Model" to SC16's parent technical committee on "Data Processing" (TC97). The recommendations were finally adopted by TC97 at the end of 1979 as the basis for the development of standards for OSI within ISO.

The reference model does not apply directly to products or equipment; only to a framework for development and usage of OSI standard protocols which themselves apply directly to systems. These standards then define how the external behaviour of open systems must conform to the OSI architecture; the internal organization and functioning are beyond the scope of the OSI standards because they are not visible from other interconnected systems.

The model has gained wide (although not universal) acceptance since its inception in 1978. Because of the reference model's generality and relevance in the design of network systems in the eighties, the model will be used as an example in the discussions to come.

2.3.1 General Principles of OSI Layering

One of the provisions of the reference model is for the layering or decomposition of the interactions between systems into manageable pieces. SC16 developed a comprehensive list of criteria for defining the specific set of layers to be used in the OSI architecture [Zimme80]. Some of these criteria are summarized here.

First, it is important that the layers remain independent of each other; the more localized the functions are, the less complex they become and the less effect they have on the services and interfaces between adjacent layers. This capability becomes important when the

time comes to change the layers' contents and protocols because of changing technology, changing user requirements, and the like. Second, similar functions should be collected into one layer; functions which differ in technology and in the processes they perform, should be separated into distinct layers. Third, each layer should only have interfaces with its adjacent upper and lower layers, and the boundary between layers should be chosen to keep the boundary interactions to a minimum.

Let us now take a look at the common vocabulary used to describe the discipline of OSI layering. Figure 2.3 illustrates the seven layers of the OSI architecture, and represents a network with two hosts, and one node in the subnet. The bottom layer corresponds to the interfaces with the physical media for the physical transmission of signals between interconnected systems. The upper layer corresponds to those functions performed by application programs and users of the network.

A note about the structure of Figure 2.3. The intermediate systems represent nodes in the subnet, whereas the end systems are the hosts connected to the subnet. Nodes therefore contain only the lowest three layers of the reference model; the hosts contain all seven. We thus have an additional criterion distinguishing nodes from hosts. Additionally, the lower three layers are collectively called the "lower-level", and the top four layers are collectively called the "higher-level".

The term "system" or "processor" will be used when referring to either a node or a host in the network. Specific use of the terms node and host will be made only when necessary.

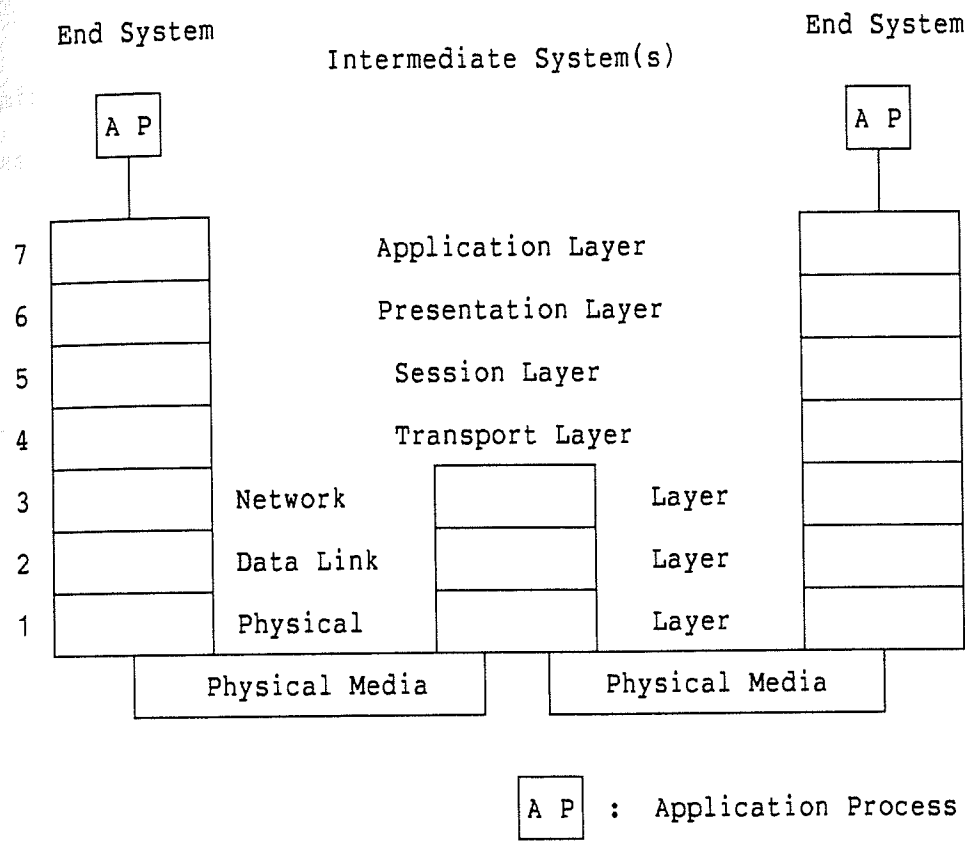


Figure 2.3: The Seven Layers of the OSI Architecture

We will refer to an arbitrary layer as the (N)-layer, its adjacent lower layer as the (N-1)-layer, and its adjacent higher layer as the (N+1)-layer. Similar notation will be used to describe all concepts related to layering (for example, entities in the (N)-layer are called (N)-entities).

With the assistance of Figure 2.4, we would like to introduce the following terms: process, entity, peer entities, parallel entities, adjacent entities, subsystem, and service-access-point. Where possible, we will use the entities numbered in the figure to give an example of the term.

A process consists of a program, its data area, its status information (that is, its program control block), and any resources (for example, a processor or communication mechanism) it needs to execute.

An entity is a process or part of a process which executes a protocol. Peer (N)-entities are two (N)-entities in different systems which communicate using an (N)-protocol (entities 6 and 7). Parallel (N)-entities are two (N)-entities occurring in the same system (1, 2, and 3), and adjacent entities are two entities communicating via an interface across adjacent layer boundaries in the same system (5 and 9, 7 and 0).

The relationship between entities and processes (which is an implementation consideration) may be one of four possibilities. First, each entity may be a separate process. Second, several parallel (N)-entities may be one or a part of one (N)-process. Third, several adjacent entities may be part of a multi-level process. Finally, their relationship may be any combination of the three.

Under the OSI reference model, each system is composed of an ordered set of subsystems, each of which corresponds to one of the layers of the network architecture. For the purposes of representing interactions between subsystems within each layer and between systems, each (N)-subsystem is viewed as being made up of all the (N)-entities for a system. For example, entities 5 and 6 form the (N)-subsystem for system A, and entities 5, 6, and 7 together form the (N)-layer.

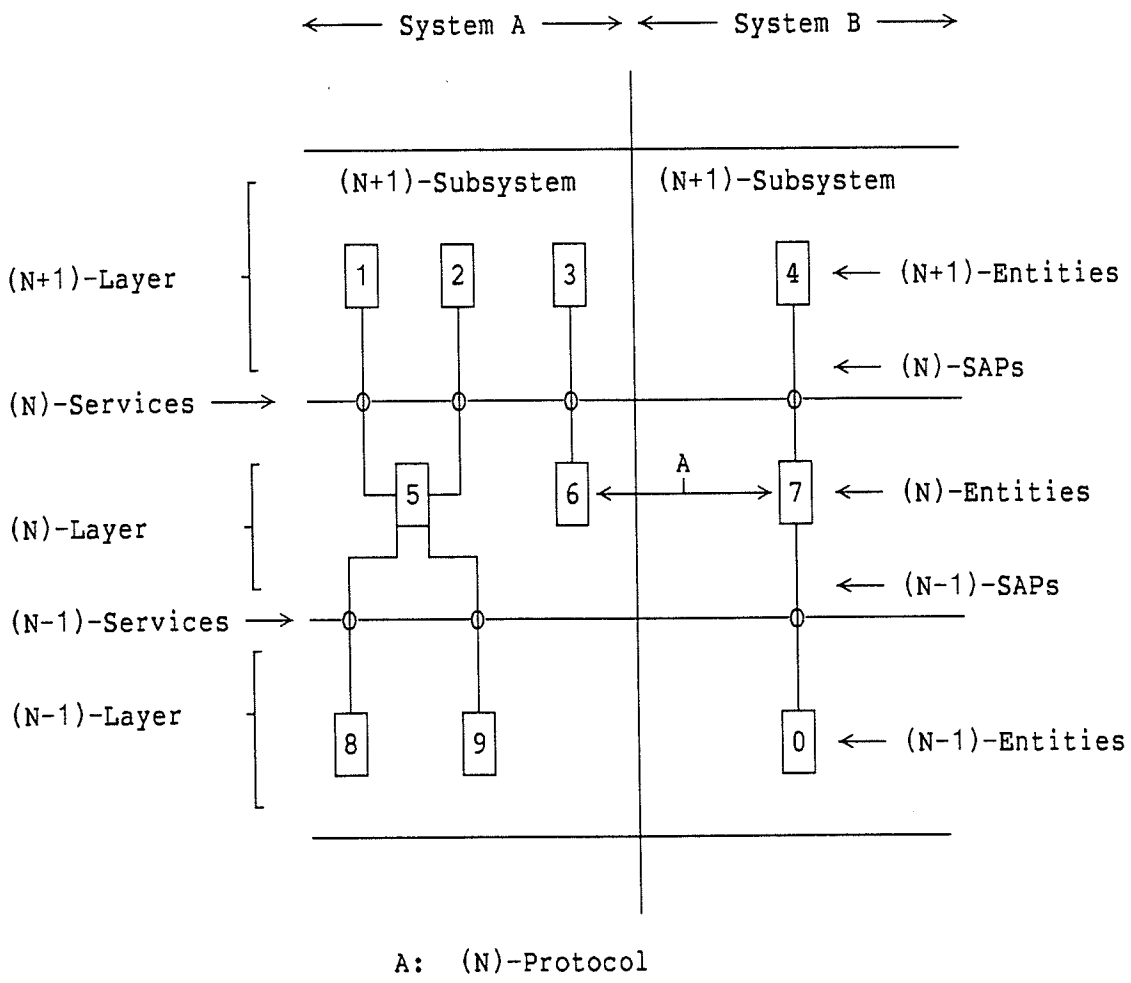


Figure 2.4: Some OSI Terminology

The boundary between the (N)-subsystem and (N+1)-subsystem in the same system is made of (N)-service-access-points (SAPs), through which one (N)-entity and one (N+1)-entity interact. That is, SAPs are the logical interfaces between entities in adjacent subsystems.

(N)-entities communicate using the (N-1)-services, and entities in the lowest layer communicate through the physical media. Direct communication between (N)-entities in the same system (for resource sharing for example) is not visible from outside of the system, and is therefore not covered by the reference model.

(N)-entities must co-operate, therefore, protocols for each layer are required. (N)-protocols describe the operation of each entity within a layer in response to commands received from the adjacent higher and lower layers, and internally initiated actions (such as timeouts).

2.3.2 OSI Architectural Layers

The following is a brief outline of some of the major functions of the seven layers of the OSI reference model. Referring to Figure 2.3, the layers are, from lowest to highest: physical, data link, network, transport, session, presentation, and application.

2.3.3 Physical Layer

The physical layer has two essential functions. First, it has the task of interfacing systems to the physical media. Second, the physical layer performs the transmission or relaying of raw bits over the media.

Taking a closer look at the physical layer, we see that the layer is concerned with the mechanical, electrical, functional, and procedural interfaces to the communication media. Issues of concern include the attributes of the physical connection devices; arrangement of pins, screws and latches; defining voltage levels used to represent digital signals; signalling rates and distances (for example, the number of microseconds occupied by a bit); error detection and correction; how the physical connection is activated, maintained, and deactivated; and whether transmission may proceed simultaneously in both directions.

2.3.4 Data Link Layer

The data link layer has one major and essential function: to transform the raw transmission facility provided by the physical layer into one which is reliable and free of transmission errors. In other words, the data link layer provides error and flow control.

Error control involves solving the problems caused by damaged, lost, or duplicate packets. Because the transmission facility may be unreliable, packets can be lost or damaged. In this case, the data link layer software on the source host must retransmit the packet. However, multiple transmissions of the same packet may introduce the possibility of duplicate packets. It is up to this layer to handle these situations so that the network layer can assume it is working with an error-free (virtual) line.

Flow control is concerned with preventing a fast transmitter from drowning or overrunning a slow receiver. Not only must each system have sufficient buffer space available, but some mechanism must be employed to indicate to the transmitter that it must stop when the receiver has insufficient buffer space left.

2.3.5 Network Layer

The network layer has two basic and essential functions. First, to control the operation of the subnet. For this reason, the network layer is sometimes called the communication subnet layer. An example function is the relaying and routing of packets (the unit of information exchanged in the network layer) within the subnet. Second, to determine the characteristics of the node-host interface.

The network layer accepts messages from the transport layer, converts them to packets, and ensures the packets are directed through nodes in the subnet to the receiving host. One issue of concern is who should make sure all packets reach their destination intact and in the proper order, the host or the subnet? As well, the network layer must determine the best method of routing packets through the network, and the best way to handle congestion control. When too many packets are present in (part of) the subnet, performance degrades. This situation is called congestion.

One of the most important design questions at this layer is the nature of the service provided to the transport layer. Two models dominate: datagram and virtual circuit.

A datagram is a finite length packet of information consisting of header and data components sent from source address to destination address, independent of any other datagram sent. A datagram service usually offers no guarantee that the messages will be delivered undamaged, unduplicated, and in the order originally sent. Delivery is performed on a "best-effort" basis.

Datagram delivery operates on a simple concept. When a datagram reaches a node, the node decides whether to keep the message, or route it to another node, based on the header destination address. If it is to be sent on, routing tables are consulted to determine which link leaving the node should be used to route the datagram. Once the datagram is sent on, the node can forget about it.

A datagram service is suited to transaction-based communication (inquiry-response type), in which the conversations are relatively short and there are varying, indeterminate delays between transmissions.

A virtual circuit is a logical channel between source and destination address (an association). All packets are guaranteed to arrive in sequence and without loss, duplication or damage. Other services, such as flow control and buffering, are also handled. The destination address is needed only during initial circuit establishment, therefore, transmission overhead is reduced. This type of service is suited for session-based conversation and bulk data transfer.

Other functions of the network layer include error control and flow control to optimize usage of transmission resources. Since this layer controls the operation of the communication subnet in general, some accounting function may be built into this layer, to count the units of data passing between hosts and produce billing information.

2.3.6 Transport Layer

The basic and essential function of the transport layer is to perform end-to-end control and optimization of the transmission of data between hosts, and is sometimes called the host-host layer. The transport layer accepts data from the session layer, splits them up into smaller pieces if necessary, passes them to the network layer, and ensures the pieces arrive correctly at the receiving host. This process must be performed efficiently and in a way which isolates the session layer from changes in hardware technology.

One function is establishing, maintaining, and releasing transport layer connections required by the session layer. For example, normally the transport layer creates one network connection for each transport connection needed by the session layer. If higher throughput is required, multiple network connections may be established by the transport layer. Data are then divided or split among these connections to obtain improved throughput. If, however, creating or maintaining a network connection is expensive, the transport layer may multiplex several transport connections onto one network connection, reducing the cost. Splitting and multiplexing are transparent to the session layer.

Also of importance is determining what type of service to provide the session layer. That is, whether to use a point-to-point error free channel, a datagram service sending isolated messages with no guarantee about order of delivery, or a multipoint (broadcast) channel. This service is determined when the connection is activated.

Other issues include the regulation of information flow between hosts so a fast host does not overrun a slow one, and the creating and deleting of connections across the network so two hosts can communicate with each other.

The transport layer is a true end-to-end layer in that protocols here are carried out between source host and destination host, as opposed to between hosts and nodes in the subnet (like in the lower three layers).

2.3.7 Session Layer

The role of the session layer is to perform functions necessary to support dialogue between programs, or more specifically, between application processes, including initialization, synchronization, and termination. This layer is indirectly the users' interface into the network.

A connection between processes is usually called a session and setting up a session is a complicated task. The two ends of the session must be authenticated to ensure the conversation can actually take place. Then they must agree on options in effect for the session (such as half or full duplex operation). Once a session has been established, it must be managed correctly. This may involve recovering from broken transport connections, ordering messages when the transport layer does not, and other such application oriented functions.

2.3.8 Presentation Layer

The role of the presentation layer is to handle the representation of information which applications wish to exchange or manipulate. Examples of this service include data compression, for example by reducing the size of files through elimination of repeated characters; encryption to provide security; conversion between character sets, such as ASCII and EBCDIC; and file conversions to handle incompatible file formats, and the like.

2.3.9 The Application Layer

The application layer is the highest layer in the OSI reference model providing the services and functions necessary for application processes in different open systems to communicate to achieve a common (distributed) task.

Whereas a lower layer provides services to the next higher layer, the application layer provides services directly to the application processes. These services are likely to be extensive, since it is anticipated that a wide range of applications will wish to become users of the network (automated office systems, banking, and so on).

Interprocess communication (process-to-process communication) is accomplished by means of application layer protocols. There are a rich variety and potentially large number of these protocols. While recognizing the broad spectrum of applications which must be supported, it is convenient to classify them and their protocols into three categories: management protocols, system protocols, and user specific protocols.

An essential part of the operation of a network is the management of the various elements and resources forming the network. To cover these aspects of networking, the OSI reference model identifies management functions as specific applications. Consequently, management protocols reside in the application layer, thus having the possibility to use all functions available from the protocols in the layers below. Two management protocols envisaged as candidates for standardization in the near future are system management protocols and application management protocols [Canad82].

System management protocols deal with the management of resources across all layers, including error control and recovery which cannot be handled by each of the lower layers, activation and deactivation of OSI resources, monitoring and reporting of status of resources across the open system, and so on.

Application layer management protocols deal with the semantics of information exchanged between application processes. This includes handling the allocation and deallocation of application process resources, detection and prevention of resource interference and deadlock, checkpoint and recovery control, and security access control.

Also in need of standardization are system protocols. Three protocols in this category are file transfer, virtual terminal, and remote job entry.

File transfer protocols are rules developed to control the various file manipulation services offered by the system, allowing the movement of large data files back and forth among a group of hosts. Major service requirements include inquiry and update file access, bulk data transmission, and data entry/data collection.

Virtual terminal protocols allow human end users to access application processes through a wide variety of terminals, and will eliminate the effects of existing incompatibilities between terminals of the same class.

Remote job entry protocols allow a human end user to submit a job to a remote host using either the job control language (JCL) of that host,

or the network-wide JCL. This will relieve the user of having to know differences between the JCL on different machines.

User specific protocols are rules designed for the manipulation of specific applications, such as banking or office automation, and are beyond the scope of OSI.

There has been some controversy and uncertainty over the layer in which the system protocols should be placed. This uncertainty causes problems when attempting to define protocols for the upper layers. In fact, some system protocols, such as a file transfer protocol, may extend across several layer boundaries. The upper layers become ill-defined, and without a rigorous description of the functions required, protocols cannot be developed on a per-layer basis. The work undertaken by committees like ISO, will greatly influence the final structure of the application environment.

A great deal of emphasis is placed on the application layer in the succeeding chapters because it is only through the services provided at this layer that integration and its full potentials can be realized.

2.4 COMPARING NETWORK ARCHITECTURES

Now that we have reviewed networking in general, and have seen the importance of the OSI reference model, it is time to examine some of the highlights of two formal networking architectures: SNA and DECNET.

IBM took its network initiative with the announcement of Systems Network Architecture (SNA) in September of 1974. By November of 1976

IBM introduced its Advanced Communication Function extension to SNA to provide networking for multiple host systems.

Not to be outdone by mainframers, Digital Equipment Corporation (DEC) developed DECNET and the Digital Network Architecture (DNA) [DEC76]. DECNET is the name for a set of software packages designed to allow the interconnection of DEC operating systems to form computer networks.

Figure 2.5 shows the approximate correspondences between the OSI reference model and these two network architectures.

<table>
<tr><th>Layer</th><th>ISO</th><th>SNA</th><th>DECNET</th></tr>
<tr><td>7</td><td>Application</td><td>End User</td><td rowspan="2">Application</td></tr>
<tr><td>6</td><td>Presentation</td><td>NAU Services</td></tr>
<tr><td rowspan="2">5</td><td rowspan="2">Session</td><td>Data Flow Control</td><td rowspan="2">(None)</td></tr>
<tr><td rowspan="3">Transmission Control</td></tr>
<tr><td rowspan="2">4</td><td rowspan="2">Transport</td><td rowspan="2">Network Services</td></tr>
<tr><td rowspan="2">Path Control</td></tr>
<tr><td>3</td><td>Network</td><td>Transport</td></tr>
<tr><td>2</td><td>Data Link</td><td>Data Link Control</td><td>Data Link Control</td></tr>
<tr><td>1</td><td>Physical</td><td>Physical</td><td>Physical</td></tr>
</table>

Figure 2.5: Comparing SNA and DECNET to OSI Layering

2.4.1 Systems Network Architecture (SNA)

Systems Network Architecture (SNA) was developed by International Business Machines (IBM) in an attempt to eliminate the chaos of

supporting several hundreds of communication products and dozens of teleprocessing access methods. The original release, in 1974, allowed only centralized networks (a tree shaped topology with many terminals connected to one host). By 1979, a more general version was released supporting multiple hosts, each with its own tree structure [Green80].

SNA primarily defines a set of communication related functions that are distributed throughout the network, and the key formats and protocols needed to tie these functions together. Reliable communication is maintained among programs and storage media which may be distributed anywhere in the network. An SNA network consists of machines called nodes, of which there are four types: Type-1, Type-2, Type-4, and Type-5 (there are no Type-3 nodes).

Type-1 nodes are the terminals. Type-2 nodes are the cluster controller nodes (CLC). A CLC node is attached to a communication controller, assists with pacing data flow for a session, and generally supervises the behaviour of terminals and other peripherals. A Type-4 node is the communication controller (COMC). It controls the communication lines and resources, and helps relieve the CPU of work associated with data communication. Finally, there are Type-5 nodes, or the hosts themselves, which handle general purpose data processing functions.

Each node contains an element of some of the SNA functional layers. Nodes are grouped together to form a control domain. A control domain can be thought of as a single tree, the root being the host, with all other nodes distributed in tree configurations. A domain exists for the

co-ordination of specific network operations, and there may be many such domains with application processes in each domain communicating with each other (a forest?).

Each node contains one or more Network Addressable Unit (NAU). A NAU consists of a set of functions that provides ports for communication through the network. The outer layer of a NAU is called NAU services and the outermost layer of NAU services, at the boundary to the application process, is the NAU services manager. Each NAU has a network address and all application processes in the SNA network interface to one of these NAU service managers.

The NAU network address is internal to SNA, is used to route messages between NAUs, and remains transparent to the application process. When two NAU service managers, each in their respective NAU, want to communicate, a session must be set up. The session defines a logical connection between two NAUs. For each pair of NAUs there exists one session, and each NAU can support multiple sessions.

There are three types of NAUs defined by SNA: system service control point (SSCP), physical unit (PU), and logical unit(LU). The SSCP is responsible for a number of network management operations, including the general management of the control domain, bringing up the network, establishment of the logical connections between NAUs, recovery and maintenance, and so on. Work is delegated between the one or more SSCPs existing in the network. Each SSCP has at least one PU companion. The physical unit helps bring a node online and offline, tests the node, and performs other similar administrative tasks. The SSCP and PU work

together to control the network configuration and data transportation resources. An LU acts as a window for the application process to access the network. That is, it is a port that application processes attach to, in order to access SSCP-provided services necessary for establishing logical connections between LUs.

A discussion of SNA would not be complete without a brief overview of its functional layers. IBM chose to organize the SNA services into layers for basically the same reasons as ISO (see Section 2.2.2). The services are divided among five layers. The layers, from highest to lowest, are NAU services, data flow control, transmission control, path control, and data link control (see Figure 2.5).

The NAU Services Layer consists of a number of outer components of the SNA network. One such component is the NAU services manager which exists for each NAU, allowing an application process to link to the system. Services for each individual NAU are called per-NAU services. An example of a per-NAU service are the network services which allow a logical connection between two or more NAUs. Network services are concerned with the activation or deactivation of parts of the network (for example, with SSCP-LU, SSCP-PU, SSCP-SSCP, and PU-PU sessions). A second component of the NAU services layer is the Function Interpreter for Function Management Data (FI.FMD). One of these exists for each session and most importantly, it involves the representation of data being exchanged. FI.FMD is a per-session service composed of presentation services to define the process connection through code, formatting, editing, compression and compaction, and session recovery services needed to handle various recovery actions.

The Data Flow Control Layer (DFC) regulates the application process' send/receive pacing and its request/response flows ("my turn, then your turn" type of control). The layer helps ensure flow integrity. Process requests, such as whether to communicate full duplex or half duplex, are also dealt with.

Transmission Control performs the co-ordination of session transmissions (sequence numbering, rate control). Transmission control also co-ordinates session establishment and disestablishment, and is commonly called the "front office" to the communication network for the application processes.

Responsibilities in the Path Control Layer include the routing of incoming data (packets) to links or to correct points in its own node, and the allowance of alternate path routing between nodes.

Data Link Control functions include the transferring of packets intact along a noisy transmission line, controlling the flow of data on the links, establishing and terminating a logical connection between stations, handling the data transfer between these stations, and ensuring message integrity in the transfers.

One difference between SNA and the OSI reference model is that SNA was designed for a homogeneous environment, where all components are designed by the same manufacturer, while OSI deals with the heterogeneous environment of diverse designs and manufacturers. While we have seen the similarities in their basic layered structure, the protocols that are used are, for the most part, different.

Another key difference between SNA and the OSI reference model is that while the reference model has an autonomous style interconnection, SNA has a hybrid autonomous-hierarchical structure. That is, SNA is hierarchical within each tree or domain (with its own SSCP), but with autonomous interconnection between trees at the level of the host-attached communication controllers. We know that in the autonomous style of interconnection, there is no master/slave distinction and there need be no identifiable central control point. SNA, however, is a centralized control architecture, and thus one of the reasons for its criticism. If a host goes down, all terminals in its domain are isolated because of the loss of the SSCP. Another host, however, may take control of the domain.

2.4.2 DECNET

Another well known computer network product is the set of DECNET offerings from Digital Equipment Corporation (DEC). DECNET is the collective name for a set of programs and protocols which extend various operating systems produced by DEC, so they may be interconnected with each other to form computer networks. The architecture used by DECNET is called the Digital Network Architecture (DNA).

As illustrated in Figure 2.5, DECNET consists of five layers (from lowest to highest): physical, data link control, transport, network services, and application. The bottom four layers correspond closely to the lowest four layers of the OSI reference model, except that DEC calls layer 3 the transport layer instead of the network layer.

The functions in the layers are distributed among the nodes. Unlike computer networks such as ARPANET which separate the hosts from the subnet, DECNET makes no distinction between hosts and communication processors. In other words, DECNET is simply a collection of machines (each of which functions as both host and node) called nodes, which may run user programs, perform switching functions, or do both.

The following is a description of each of DECNET's five layers, starting with the lowest layer.

The Physical Link Layer manages the physical transmission of bits over the communication facility. The layer is concerned with the physical characteristics of the data channel, signalling techniques, clocking on the channel, and the interface between the computer system and the communication services. The physical layer can be implemented in the operating system itself, or as a separate front end processor.

The Data Link Layer creates an error free communication path between adjacent nodes so that data blocks may be transmitted reliably between them. In addition, the data link layer manages transmission and reception on various types of channels (half duplex for example). This level is channel independent.

The Transport Layer transports messages from source node to destination node using various routing schemes. The establishment of a path between two end nodes is also performed at this level, through the use of the individual node-to-node paths in the data link control layer below. Various routing tables and routing algorithms are used to switch messages from an incoming channel to an outgoing channel. Network congestion is also controlled here.

The Network Services level handles the actual end-to-end creation and management of a session between two nodes. It allows data to be exchanged reliably and sequentially, independent of the node's location in the network or which network systems are communicating. Other network services include flow control (pacing), buffer management, data integrity checking, interrupts, and control of addressing.

User application programs, resource managing programs, and function oriented programs all run at the Application Layer. This layer supports the exchange of data between application programs and input/output devices or file systems, and the simultaneous execution of programs using independent logical links. Application services handle resource sharing of local and remote peripherals, file transfers between one system and another, remote file access allowing file manipulation on a remote system, data base management providing access to distributed data at a simple level, and network management.

DECNET has no separate session layer and its application layer is a combination of ISO's presentation and application layers. DECNET does not allow data transformations such as text compression and encryption, although certain other transformations provided by a file access protocol take place while transferring files from one system to another.

Chapter III

DISTRIBUTED SYSTEMS

The overall goal of network architecture design is to meld the diversity of resources and protocols into some coherent form so that the collective human/software/hardware resources can intercommunicate. The common current approach is to create a transport or end-to-end protocol built on lower-level protocols that move uninterpreted blocks of data, and then construct, on top of this service, protocols for whole-file transfer, virtual terminal access, and other virtual device usage.

However, the full potential of computer networking for resource sharing and distributed computing cannot be realized with such an architecture for the following reasons. First, each programmer wishing to provide or use a new network-sharable resource must face anew all the issues of data-type translation, naming, protection, and interfacing to the transport layer. Second, a terminal user must know the various naming and other access mechanisms required by the network, each host, and each service. He must login to his local host, use a network access program, and then login to his target host(s), each probably using different conventions. And third, the setting up of accounts and other administrative procedures are awkward. The user must explicitly establish accounts and receive billing from each administration controlling a host with resources he wishes to use.

The next step for the elimination or minimization of these and other problems is the creation of a distributed system. This step is a giant leap in computer network design that better allows us to exploit the potentials for achieving the envisioned benefits of an integrated environment - such as reduced user interface complexity, better control over information flow, and increased system efficiency - and an improved understanding of the functional requirements and operational behaviour of network systems.

Attempting to derive a concise definition of a distributed system is a difficult task. The technical and commercial literature is full of definitions which contradict each other and which, in many cases, are so vague that it does not mean anything to adopt or to reject them. For our purposes, a simple definition will do.

A Distributed System is a computer network with a high degree of cohesion, transparency, and decentralization.

Cohesion is the tendency to remain united and well-knit, therefore, a distributed system is something more than a cluster of computers that act independently. Transparency means viewing the distributed system as a nondistributed system at some level. In other words, the user of a distributed system should not be aware that there are multiple processors, or that his computations and data are dispersed among various physical locations, unless he wants to be. Decentralization implies that decisions are made at different physical locations in the system. Thus, a computer network must exhibit cohesion, transparency, and decentralization in order for it to be called a distributed system.

We will now attempt to identify what are believed to be the most interesting and important characteristics of distributed systems, in order to strengthen our definition above. We have selected three criteria which seem to constitute the most widely accepted characteristics: multiplicity, existence of a high-level operating system, and system transparency.

A desirable (but not necessary) characteristic of a distributed system is a multiplicity of assignable resources to provide services within the system. The modifier "general-purpose" has often been used to describe these "assignable" resources, such as general-purpose computer systems. However, it is not clear that this modifier is either necessary or sufficient. What is necessary is that the system have the ability to be dynamically reconfigured on a short-term basis with respect to those resources that provide specific services at any given time [Enslo78]. Resources not directly involved with the reconfiguration should not be affected. As an example, if the system utilizes a general-purpose processor in its daily activities, then there may be many such processors. A system usually consists of general-purpose resources and special, fixed function resources (possibly being dedicated, nonreassignable, or both). The availability of such resources leads to high system reliability and availability with graceful degradation, and characterizes a flexible and expandable system.

A high-level operating system (or system-wide operating system) that unifies and integrates the control of the distributed components is one of the most important and desirable characteristics of a distributed

system. In other words, system software makes the difference between a cluster of computers that act independently and that same cluster integrated into a smooth-working, cost-effective distributed operation. There can be no hierarchical relationship between the high-level operating system and all of the local (host) operating systems, since the criterion of autonomous operation would be violated. The local operating systems need not be homogeneous either, although heterogeneity complicates the system design problem considerably.

System transparency allows the user to connect to the system with the impression of having one "super-system" at his fingertips, rather than a collection of heterogeneous systems. The user need not specify which physical or logical component is needed to perform a specific function. The system should be totally transparent to the user, unless he specifically requires knowledge of the system configuration in order to accomplish a certain task. For example, the user would usually request services by name rather than by designation of the resource, yet if the user knows which resource would best execute his task, he should be able to get it. The user has the ability to develop programs and routines as easily as if he were working with one centralized system. It is up to the system-wide operating system to deal with all varieties of command languages and data definitions present in the system. This is a non-trivial task.

Since a distributed system is a special case of a computer network, other characteristics we discussed in Chapter 2, such as autonomy, and the physical distribution of the physical and logical components of a computer network interacting through a communication subnet, are

certainly an integral part of distributed systems. One characteristic not explicitly mentioned in the definitions of computer network and distributed system, but which is never-the-less implied, is the existence of disjoint address spaces (no shared memory) for each host. This characteristic then excludes systems with multiple processors communicating through shared memory from our definition (that is, a single host).

We now have a set of properties and operating characteristics which, when combined together, uniquely characterize a distributed system and further distinguish it from a computer network. For example, a computer network is distributed in the sense that its components (hosts and nodes) are physically distributed, and yet it is not cohesive, transparent, or decentralized. Systems such as these do not have the ability for dynamic relocation or reassignment of tasks in the event of hardware or software failure.

There are three activities of a system that may be distributed: data, computation, and control (such as the operating system). In order to achieve an appreciable portion of the benefits of distributed systems as outlined in our definition, it is essential that the system exhibit a high degree of distribution for all three of these activities.

Distributing computation is the distribution of processing by dividing a single application into pieces and running the pieces on different machines. Computation may be distributed to achieve functional separation (different functions are performed by different machines), closeness according to some measure, a given level of fault

tolerance, or other such goals [Lamps81]. As an example of functional separation, activity supporting data base activity may run on one set of hosts, while activities supporting extensive numerical computations may run on another set of hosts.

Distributing data is the distribution and organization of data among the system hosts. Some files may be replicated at each host, while others may be split among several hosts. The decisions on how to organize the data will depend on costs (in terms of money, reliability, responsiveness, and so on) and on available mechanisms to support consistency of multiple copies, maintain synchronization of access, and so forth.

Distributing control is the distribution of resource management and consists of effecting the decisions and actions involved in regulating access to a resource, as well as implementing the resource data structures and operations visible to its users. This means that the system state representation is partitioned and scattered all over the different processing elements (hosts and nodes). The control of all executive functions normally done within processing elements must be done among processing elements.

Each of these issues are common to many applications and can be discussed in general terms at the application layer. They will prove to be useful for our discussion of integrated distributed systems in Chapter 5. Therefore, the following three sections will elaborate on each of these issues.

3.1 DISTRIBUTED DATA

One of the most important considerations in the design of distributed systems is the distribution and organization of data among the system hosts.

Data, distributed or otherwise, can be kept in one of two forms: in a data base or in a collection of files. The arguments in this section relate to either form.

There are a number of ways in which data can be distributed and used. The following outlines the types of data system configurations we will describe.

- hierarchical distribution
  - dependent hierarchical data
  - independent hierarchical data
- partitioned (or split data) distribution
- replicated distribution

One way to distribute data is hierarchically, in which the data are distributed between two or more hosts in some hierarchical fashion. With dependent hierarchical data, the data in the lower-level hosts are closely related to those in the higher-level host. They are often a subset of the higher-level data used for local application. The master copy of the data may be kept by the higher-level host. When a change is made to the data in the lower-level host, this change must be passed up to the higher-level host - sometimes immediately, sometimes later in an updating cycle. For example, the higher-level host might store customer numbers, names, credit information, and details of orders. These are also stored by the lower-level hosts, and any modifications to them must

be passed upwards. With independent hierarchical data, the structure of the data in the lower-level hosts is probably different and independent from that in the higher-level host. A common example of such a relationship is one in which the lower-level hosts are designed for routine repetitive operations such as order entry, production control, inventory, and so forth. The higher-level host is an information system, possibly at a head office location, designed to answer requests from management, forecasters and so on. The higher-level host then accesses the data from the lower-level hosts and summarizes, edits, and reorganizes them to answer the requests.

Another way of distributing data is called partitioned distribution or split data distribution. Here there are multiple hosts containing identical data structures and formats, but not identical data. Host A keeps district A data, host B keeps district B data, and so on. Most of the transactions processed require the data be in the host which handles them, but occasionally a transaction originating in one host needs the data in another host. Either the transaction or the data must be transmitted over the subnet.

Data may also be replicated among the system hosts. Identical copies of the data are stored in separate hosts, because this duplicate storage avoids the need for high-volume transmission between the hosts. Such an organization only makes sense if the volume of updates of the data is low. An example would be a public data service such as the British Post Office Prestel system. This makes data available on home or office television sets linked by telephone lines to the system. Multiple copies of the same data will be stored on relatively small local systems, and be updated from a central system.

Let us now turn our discussion to some of the considerations which affect the location of data in a distributed system.

Data for a particular application in a distributed system may be either centralized or distributed. There are properties inherent in certain data which lead naturally to distribution, and properties in other data which lead naturally to centralization.

The main property favouring distribution is that the data are used at one location and are rarely or never used at other locations. Much of the information in a branch office (client addresses for example) is of no use anywhere but at that branch office. However, other information generated in a branch office may be needed elsewhere (such as customer orders which are needed in manufacturing plants).

A property of data which argues strongly for centralization is that the data are being constantly updated and referred to by multiple users in different geographical locations. The users need an up-to-the-minute picture of the data as a whole, and the data are being modified by users in different locations. One copy of the data is therefore kept in one place. This is done on reservation systems for airlines, hotels, and rented cars.

Data to which many inquiries are made could be distributed if the data are updated only infrequently, if the data given to the inquirers can be a few hours old rather than a few seconds, or, possibly, if the updates come from only one source.

Distributed data can have a number of problems associated with them. The problems are such that it is often desirable not to have an unconstrained distribution and replication of data. The problems we wish to discuss briefly are response time, multiple copies of data, and deadlock.

Distributed data imposes a response time component on each request that cannot be satisfied locally. This component's two parts are the time spent waiting for service by the subnet, and the time for the subnet to deliver the message. Many transactions will require more than one message transmittal. A well designed subnet must minimize communication time so that responses to remote requests will not take significantly longer than responses to local requests. A poorly designed subnet will provide unacceptable response times because of excessive communication delays.

If hosts in more than one location need to use the same data, it is possible that they could each keep their own replicated copy. There are several possible advantages to having more than one copy. First, the reduction in communication costs may be greater than the increase in storage and possibly processing costs. Second, for distributed systems with many copies of certain data, if one copy is damaged, a duplicate copy exists. Third, system availability as perceived by the users may be enhanced. Fourth, response time to users may be improved.

The main disadvantage of having more than one copy, apart from storage costs, is that updating and file reorganization have to be done multiple times. If the data are never updated, this presents no

difficulties. The system designer would be concerned mainly with costs. He would determine whether it is cheaper to have multiple copies or to use the network to access one common copy. Generally speaking, the more frequent the updates, the less desirable data replication is.

A deadlock occurs if two or more processes bid for the same resources in a mutually exclusive manner. Consider the following example. Process A starts and seizes and locks out file A while process B starts and seizes and locks out file B. Process A then requests file B while process B requests file A. However, both files are locked out. Hence, neither process can complete and release its file, causing an indefinite blind alley. This situation must either be prevented or detected to avoid tying up the entire system.

It would seem that this problem could be aggravated with distributed data because of the time delays involved in remote file updates. A solution may be quite complex, because no one host may be totally aware of what processes have acquired what resources.

There are other significant factors that must be considered in distributing data, several of which are discussed in [Liebo81] and [Marti81b]. These include data security, error recovery and reliability, update interference issues and so on, and are beyond the scope of this thesis.

3.2 DISTRIBUTED COMPUTATION

Distributing computation is the distribution of processing by dividing a single application into pieces and running the pieces on different machines.

Two types of distributed computation have been defined: horizontal (or flat) and hierarchical (or vertical).

A distributed computation design in which all processors operate at a logically equal level to perform their tasks, is called a horizontal configuration. Note the term "logically"; the processors may be physically unlike and of different capacity and power. The important aspect is their logical relationship within the distributed system.

A good example is a configuration of three hosts geographically distributed and interconnected via a communication subnet (ARPANET for example). Here, each host will normally handle jobs/transactions which originate locally. The interconnections can be used for load levelling among the three hosts. Candidates for load levelling could range all the way from compilations to complex jobs which require that the referenced files be transferred between cities with the jobs.

With vertical distribution the interconnected processors form a hierarchy, sharing tasks in a structured way. Each level is usually more sophisticated than its predecessor with each component to some degree controlled by the higher level member(s) of the hierarchy. The basic design rule is to distribute the processing load within the hierarchy so that processing can be performed with an optimum

cost/performance ratio. If a function is needed quite frequently and quick response time is preferred, the function is moved down into the hierarchy. Those functions not as critical tend to migrate up towards the centre of the hierarchy.

A good example of a hierarchical configuration consists of one or more host information processors and several satellite processors placed in factory locations or bank branches, depending on the type of application. At the lowest level are the device controllers (usually intelligent) supplying functions needed for terminal control or real-time control. The host, at the top level, provides services to terminal users and satellite processors (compilation, linking, down-line loading of programs to satellites, and so on). The satellite processors depend on the host to some degree because they cannot perform these services themselves, although a full master/slave relationship does not exist since this would decrease the reliability of the system, and by our definition, would not be a distributed system.

Because of the mechanisms built into software or systems architecture, designers sometimes try to make all configurations for distributed computation vertical, or all configurations horizontal. This can result in excessive overhead, system inflexibility, or awkward control. Whether or not computation should be distributed vertically, horizontally, or both, depends on the patterns of work the configuration must accomplish and the patterns of data usage.

In deciding which particular distributed computing configuration(s) is needed we are concerned with such questions as: where are the units

of processing work required; how large are these units; what size of processing machine do they need; are the units independent, or does one depend on the results of another; what stored data do the work units employ; do they share common data, or are the data independent; what transactions must pass between one unit and another; what are the patterns of transaction flow; must transactions pass between the units of work immediately, or is a delay acceptable; what is the cost of delay [Marti81b].

The answers to these questions differ from one organization to another. The patterns of work are different. The patterns of information flow between work units are different. Different types of corporations tend, therefore, to have their own natural shapes for distributed computation. What is best for an airline is not necessarily best for an automated office system.

The decision to run a particular part of a computation on one machine or another is based on the suitability of the machines, the relative costs of the machines, the bandwidth of the communication channels interconnecting them, and the current workload. For example, consider a system in which all the modules (procedures, programs, and so on) are available on both a central machine and one minicomputer. When the workload on the central machine is light, most modules run faster there because the machine is faster. On the other hand, when the central machine is heavily loaded, the minicomputer may actually have a better response time. The question is, which parts of the program should be run where.

Consider two approaches to load balancing (that is, optimally assigning tasks to processors) among processors, the adaptive approach and the dedicated function approach, both of which are studied as part of the workload partitioning problem in general [Liebo81].

The adaptive approach allows load sharing between processors so that an underloaded processor may help out an overloaded processor with some of its tasks. Booth describes this type of dynamic load levelling between a process and its data [Booth76]. Assuming a string of information processors have the same capabilities, the process may be moved to the data, or the data may be moved to the process. In a batch environment, jobs tend to do a lot of processing against reasonably large data bases. In this case, it is more feasible to transfer the job, or some part of it, to the data base. If, on the other hand, a request using a small part of the data base is being processed, it may be wiser to move the data to the request, returning only updates to the computer where the data are stored. In general, this scheme is associated with horizontally distributed configurations.

The dedicated function approach does not permit dynamic load sharing. All functions and data base segments are preassigned to processors statically. This can lead to imbalances in processing loads at erratic intervals, although control of the computation is somewhat less complex than in the preceding case. This method is most often used with hierarchically distributed configurations.

Gylys and Edwards attempt to define a mathematical model for such a class of problems [Gylys76]. In their paper they discuss computational

techniques for optimal workload partitioning in a network environment. The biggest factor or obstacle in optimally assigning programs to computers is the communication subnet. The subnet is the least reliable part of the system, and is a potential traffic bottleneck. Messages may be lost or delayed, and communication among nodes may completely fail due to hardware or software errors. Faults of this type may be catastrophic to many real-time systems. Gylys and Edwards look seriously at minimizing subnet traffic when partitioning programs to processors. They believe that if subnet traffic can be minimized, finding optimal network configurations for a given software design follows naturally.

Stone and Bokhari describe an algorithm for finding optimal assignments of program modules across a two-processor system [Stone78]. By taking into account the real running time of each module on each of the processors and the communication time between any two modules running on separate processors, an "optimal assignment" graph can be constructed that will indicate which modules should be assigned to which processor. For a general K-processor system, an optimal solution to the module assignment problem has not been found. Wu and Lui attempt to generalize a solution for the K-processor system in [Wu80].

We have compiled a list of three reasons for distributing computation over multiple machines: reliability, modularity, and program size.

First, overall system reliability can be improved. If the application is distributed over several hosts in the system, failure of one host need not result in failure of the entire application. If a

central mainframe fails, all computation must cease, and jobs with high availability requirements (that is, those jobs used frequently by many hosts) suffer.

Second, smaller more modular application programs are possible. By segmenting parts of a program into smaller independent pieces, each running on its own processor, under certain conditions programming costs can be reduced, debugging is easier, and productivity will increase.

Third, the process may be too large and complex to run on a single computer. A good candidate for this is an application that requires several peripheral devices, each with a fair amount of processing and real-time response needs. Applications such as message switching, weather processing, and device monitoring fall into this category.

One cannot reap the benefits of distributed computing without running into some problems, two of which are system inefficiencies, and the additional complexity of interprocessor bugs.

Distributing computation may introduce certain system inefficiencies. Minicomputers and microcomputers have limited address space, therefore, memory mapping is required to maintain large physical memories. This could reduce the power of the processor by five to twenty percent, depending on the type of processor and the type of application [Liebo81]. In addition, the more messages passed between processors, the higher the operating system overhead, and the lower the power of the processor.

Another problem is the additional complexity of interprocessor "bugs". Symptoms may appear in one processor, but the problem may exist in another. Errors propagate through the system, therefore, different processors have different levels of problems. This phenomenon increases as the tightness of interprocessor coupling increases.

The bottom line is that more attention must be made to software development. More complex software is needed to build and control applications in a distributed computing environment than with a central system. As we shall see shortly, high-level operating systems represent a major contribution to the increased software development.

3.3 DISTRIBUTED CONTROL

Distributed control is the distribution of resource management and consists of the decisions and actions involved in regulating access to a resource, as well as implementing the resource data structures and operations visible to its users.

Control contains two elements: data (including control data) required to make decisions and algorithms required to manipulate data to make decisions.

In the most centralized kind of control design, data and algorithms are placed at a single point and only a single system processor has access to them. A pure master/slave relationship exists between the processor which can operate on control data and the other smaller processors. These smaller processors undertake tasks solely as permitted or directed by the central processor. This is the usual

perception of a single processing system built around a uniprocessor central processing unit.

Control may become more distributed by permitting more than one processor to access the data and algorithms which are collected at a central point. This concept is common to multiprocessor designs. The operating system has data required for system control and the algorithms required to manipulate the data. Any processor in the system is permitted to access the control structure. This is possible because the memory locations which hold the operating system are equally accessible to all processors.

A further distribution of control becomes possible by partitioning the concept of control into discrete control functions. Thus control may be partitioned into memory management, program management, input/output management, and so forth. This partitioning is accomplished by restructuring the programs of the operating system into a set of specialized service elements. When the operating system is so restructured, it is possible to allow all processors to address any control element.

In the purest form of distributed control all data and all control algorithms are dispersed throughout the system and replicated at all points. There is no global data at any point in the system. Each processor has a complete operating system which is responsible for local control and for interacting with other processors to achieve some form of system-wide control.

From the above, we see that various levels of distributed control can be provided by a high-level operating system. In our discussion of the definition of distributed systems, we stated that a high-level operating system which unifies and integrates the control of the distributed components is one of the most important and desirable characteristics of a distributed system.

A High-Level Operating System (HLOS) is a collection of software and associated protocols that allow a set of autonomous computers, which are interconnected by a computer network, to be used together in a convenient and cost-effective manner.

An HLOS must perform two functions. First, it must turn a collection of hardware/software resources into a coherent set of resources (such as processes, files, directories, clocks, and accounts) and support their naming, access, sharing, protection, synchronization, and intercommunication (including error recovery). Second, it must multiplex and allocate these resources among many computations.

In performing the functions outlined in the definition above, an HLOS must deal with the problems arising from its distributed nature and the heterogeneous systems on which it is based. Specific problem areas include translation of various data representations, distributed service and resource structures, potentially more complex error recovery, update of multiple copies of files or data bases, control by several administrations, and inefficiency arising from distance and bandwidth limitations between components [Watso80].

There are two main approaches to designing and implementing an HLOS. The first approach is to build the HLOS on top of existing operating systems. This approach is taken by the National Software Works (NSW), and is often referred to as a network operating system (NOS). The second approach is to build the HLOS from scratch as the native system on each host computer. This approach is taken by Rochester's Intelligent Gateway (RIG) and is often referred to as a distributed operating system (DOS).

There are three primary HLOS design goals. The first goal is that a process, terminal user, or programmer have a uniform, coherent view of distributed resources. Processes, programmers, and terminal users should not have to explicitly recognize whether a needed resource is local or remote. A user should not have to (although he may) program differently or use different terminal procedures depending on resource location; this implies that network operations and the idiosyncrasies of local hosts can be largely or completely hidden. A second goal is that it be possible to implement and use the HLOS structure efficiently as the base (native) operating system, and as a "guest" layer on existing operating systems that support appropriate interprocessor communication. A third goal is extensibility. The term implies that users can easily build new services on existing ones without requiring system programmers to add new resident or privileged code. Thus, an HLOS does not have to spring full-blown into existence to be useful; it can start with a few services and evolve from there.

A key design issue in an HLOS is the interprocess communication (IPC) mechanism. This mechanism is the highest layer of the communication

service, and provides the process-to-process service in the distributed system. Its purpose is to support the reliable communication of uninterpreted, arbitrary length, logical messages between end-to-end addresses.

A major concern at this level is how two processes communicate with one another. What are the semantic primitives? Two fundamentally different schemes exist: message passing and procedure calls. We will now look briefly at each of them.

When procedure calls are used as the IPC mechanism, the complete system consists of a collection of procedures written in some language. The code for the procedures is distributed among the processors. To achieve communication across machines, a procedure on one machine can call a procedure on another machine. For example, the calling procedure might contain the statement y=f(x). This calls the procedure "f" with parameter "x", putting the result in "y". The semantics are the same as for ordinary (local) procedure calls: the caller is suspended until the callee has finished, the parameter is passed from the caller to the callee, and the result is passed from the callee to the caller. If the caller and callee do not share any common address space, the parameter passing will normally be call-by-value rather than call-by-reference.

If, however, the caller and callee do share common address space, the parameter passing can be call-by-reference. In this case, information can also be transmitted by changing global variables. Co-operation among processes is achieved using locks, semaphores, monitors, and other synchronization data structures. A monitor is a collection of

procedures and data which has a lock associated with it in order to prevent more than one process from being active at any one time. Procedures accessing critical tables are grouped in this way to ensure exclusive access.

This model of IPC leads to having the entire operating system being written as one large program, which gives coherence but also lack of flexibility.

The alternative model, message passing, consists of processes communicating with one another by exchanging messages, in the form of requests and replies. Each process has primitive operations to send, receive, and wait for messages. The code for each process is conceptually separate from all the other ones; each one may be written in a different language, for example. Any restrictions about which process can communicate with which other process have to be enforced dynamically. With procedure calling, the restrictions can be enforced in the language by defining the domain of the system in which a process can be accessed (that is, its scope).

Message passing has many more issues to be resolved than procedure calling. Some of the major ones are port naming and access, blocking versus nonblocking, virtual circuits versus datagrams, and flow control and buffering. Many of these points have arisen in other protocol layers. For this reason, we will only discuss the first two of these issues.

When a process wants to communicate with another process, it must somehow indicate which process it wants to communicate with. This can

be done by either giving the other process' name or the name of a port or socket to which the other process is connected. If the space of process or port names is sparse and encrypted, possession of a valid name can be taken as evidence of permission to communicate. Using process names rather than ports for communicating effectively restricts each process to a single port.

Another issue we have not dealt with yet is that of blocking primitives versus nonblocking primitives. Suppose we have two primitives:

```
SEND(id, buffer, bytes)
RECEIVE(id, buffer, bytes)
```

If SEND is blocking, it sends the contents of the buffer to the process, port, or virtual circuit specified by "id" and waits until the transfer has completed before continuing execution with the next statement. "Completed" may mean any one of several things, depending on the system: the operating system has copied the message to its own buffers, the message has actually been sent, or the message has been sent and acknowledged.

A blocking RECEIVE does not return to the calling process until a message has arrived. An interesting question is whether a blocking primitive should block forever if the remote process is down. Little is known about the interaction of blocking primitives and timeouts.

Nonblocking primitives have different semantics from blocking primitives. A nonblocking SEND returns to the calling process

immediately, before any transmission has started. Similarly, a nonblocking RECEIVE merely indicates willingness on the part of the process to accept a message. In general, RECEIVE returns to the calling process before a message has been put into its input buffer. The actual completion of the SEND or RECEIVE happens later. The process can be notified by an interrupt or it can be provided with a primitive STATUS(id), which returns the appropriate status. Sometimes a primitive WAIT(id) is provided to allow blocking primitives to be built up from nonblocking ones.

There is a one-to-one mapping of primitives in a message passing system with primitives in a procedure calling system. For example, the SENDMSG:AWAITREPLY primitive for immediate message passing has, as its mirror image, a simple procedure call on the procedure calling system. Similarly, a SENDREPLY is equivalent to a RETURN(from procedure). Lauer cites this mapping and goes as far as saying that neither model of process communication is preferable (that is, the two models are equivalent) [Lauer79]. The major criteria, Lauer suggests, for choosing one over the other should be based on the system architecture implemented and not on the applications which will eventually run on the system.

Message passing has been used in a number of operating systems (see [Ball79]). An analysis of the differences between such schemes and those based on procedure calls was made by Lauer and Needham and discussed briefly above. At this time we do not favour one over the other, and feel that either method of process communication is viable.

3.3.1 Network Operating Systems

A Network Operating System (NOS) is a high-level operating system built on top of existing operating systems.

This means that each host continues to run its old operating system, with the NOS implemented as a collection of user programs running on the various hosts. Elegant it is not, but this approach is easy to implement and takes advantages of proven system and application software.

One way to superimpose a NOS on top of existing heterogeneous hosts is to provide each user with a process, or agent, whose job is to provide a uniform interface to all hosts. The agent may run on one of the hosts or on a separate network access machine (NAM). It may or may not attempt to hide the existence of multiple hosts from the user. If it does not attempt to hide the network from the user, typical commands to the agent are of the form: AT HOST X DO COMMAND Y. If it does attempt to hide the network from the user, the agent must select the appropriate host to run each command on.

The agent maintains a data base containing information about the hosts and information about the user's data and programs. For example, there might be an ACCOUNTING file to keep track of the user's account number on each host, his CPU budget, his disk quota, and whatever other information is needed to login and use the host. Another file could be used to protect the user from the different file naming conventions on different hosts. By creating a virtual network-wide file system, the

user only has to learn one set of naming conventions, those used by the agent. The agent uses the data base to locate files and translate between virtual file names and (host,real-file-name) pairs.

In the simplest form, the agent simply acts as a command processor, translating the user's commands to the command language of the appropriate host and then sending them there for execution. Before each program starts running, the agent must ensure that all needed files are available where they are needed. When this strategy is used, implementing the network operating system comes down to two things: designing the network command language, and implementing the agent.

This strategy is attractive because it is simple and does not affect the hosts. In fact, the hosts may not even be aware of the existence of the network. Unfortunately, its utility is limited to situations in which all needed files are known in advance, so the agent can have them sent to the proper host before the program starts executing. Furthermore, it is often difficult to arrange for interactive input and output without the program being aware of the existence of the network. A more general strategy is to intercept all processes (a user command for example), catching all of its system calls, and making appropriate requests to other components as necessary to handle the process in the context of the entire system.

A project which does attempt to use this more general strategy is the National Software Works (NSW), which runs on the ARPANET. The aim of NSW was to design and implement a NOS which supports the software development process by means of providing access to a number of tools

(editors, compilers, simulators, file systems and debuggers, and so on) [Forsd78]. NSW attempts to integrate all these tools into a "unified tool kit" under a "single monitor" with a "single file system", and follows the rule that distribution of the network resources and operations should be invisible to the user (system transparency) in order to simplify the use of a tool, regardless of the host providing that tool.

The NSW system is composed of a number of processes providing network services. The processes are the front end (FE), works manager (WM), tool-bearing host foreman (FM) and the file package (FP).

The user's interface to the network is the front end (FE). Each user logging on to NSW through his terminal has a dedicated FE process. The FE provides a single interface, command language and file system, independent of the hosts on the system. The FE also implements a virtual terminal protocol (VTP) to interface various types of interactive terminals to the system so all terminals look alike to the NSW components.

The central component of NSW is the works manager (WM), or NSW monitor. The WM is designed to satisfy a number of external requirements. First, it should support large scale operations with respect to the number of concurrent users. Many hundreds of concurrent users are considered, resulting in a large amount of storage space to be managed for administrative purpose. Second, if large scale operation is to be supported, system failures are likely to be catastrophic and expensive. This requires fault tolerant behaviour of the WM (that is,

component failures may only result in performance degradation and affect only a limited number of users). Third, since NSW is designed to be implemented on existing host computers and local operating systems, no special hardware to support reliability can be used (modifications to local OSs should be restricted to adding privileged code only).

Reliability and large scale operation require distributed control (that is, distribution of the WM and its associated data bases for cataloging files, recording access rights, accounting, and so on). If there are multiple instances of the WM to achieve distributed control, then each instance of the WM has to maintain a consistent view of the global system state with respect to resource allocation, tool operation, access rights, accounting, and so forth. Special algorithms are then needed to handle the problems associated with multiple copy updating.

The tool-bearing host foreman (FM) is the tool's interface to NSW. When the user needs a tool, the FE informs the WM which then checks its data base to locate the specified tool. An FM process is then created to provide an execution environment for the tool and to control its operation. The foreman must catch all system calls made by the tool, so that they can be carried out in the context of the NSW environment.

The file package (FP) is the file handling facility for NSW. The NSW file system as a part of the tool environment is a means to make the output of one tool available as an input to another tool, regardless of the tool location. Thus, the primary function of the file package is the creation of copies of NSW files which are suitable input to tools. A secondary function of the FP is to import external files into the NSW

file system and vice versa and to take care of peripheral operations like reading and writing tapes.

To understand the role of the above NSW processes better, consider the scenario for a tool issuing an "open file" operation. The tool-bearing host foreman intercepts the request from the tool and forwards it to the works manager, who locates the file from the file catalog. The file may have to be transported to the FM host and possibly translated to a form usable by that host. In this case, the WM must create a new physical copy of the NSW file, and calls upon the FP to do so. Two FP processes are created: a "receiver" FP on the destination host and a "sender" FP at the donor host. Once the transfer is complete, the FP processes are deallocated and once the WM tells the FM process that the file is now available, it too can be freed.

The glue holding together all of the above NSW components is the IPC facility, called MSG. It is the responsibility of MSG to provide communication between the various NSW-system processes implementing front end, foreman, file package, and works manager, and to allocate and activate processes of an appropriate class like FP and WM to support a specific user initiated tool operation. A number of process interactions are illustrated in Figure 3.1

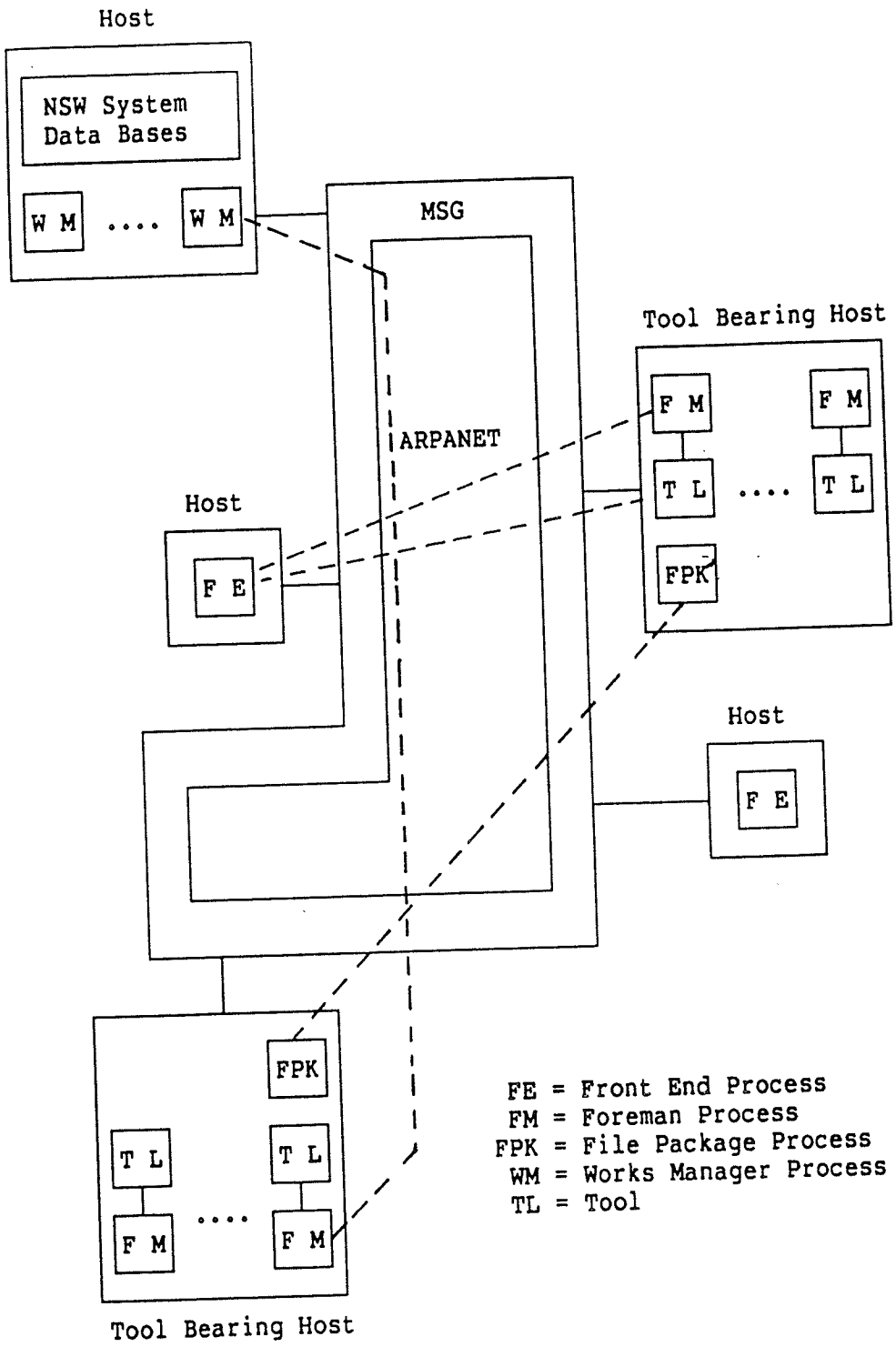


Figure 3.1: NSW Components and their Interaction

3.3.2 Distributed Operating Systems

A Distributed Operating System (DOS) is a high-level operating system built as the native system on each host computer.

This approach is to literally "throw away" the existing operating systems and start all over again with a single homogeneous distributed operating system. The rationale behind this approach is that having a single network-wide operating system is better than tying together a collection of incompatible ones in some ad hoc manner. That is why people interested in building distributed systems from scratch tend to use the DOS approach to network-wide distributed control.

Most researchers use one of two models to implement their native DOS: the object model or the process model.

In the object model, the user has access to objects or resources, each of which has a type, a representation, and a set of operations that can be performed on it. To carry out an operation on an object (read from a file for example), a user process must possess a capability (permission) for the object. The basic task of the operating system here is to manage capabilities and to allow operations to be carried out. Capabilities must be passed around in such a way that malicious users cannot fabricate capabilities at will (see [Almes82] and [Lazow81]).

In the process model, each resource (file, disk, peripheral, and so on) is managed by some process, and all the operating system does is

manage IPC. Traditional operating system services, such as file systems, processor scheduling, and terminal control, are managed by specific server processes that can be requested to perform the service. In many cases, the servers can run as ordinary user processes.

The example we are about to present actually implements both of the above models. Rochester's Intelligent Gateway (RIG) is a multiple-machine, multiple-network distributed system designed for a collection of heterogeneous machines, interconnected by networks of varying characteristics. It represents one of the earliest and most comprehensive attempts at designing a DOS and supporting network architecture from the ground up [Lantz82].

The RIG environment consists of a collection of resources (or objects), accessible by clients, and managed by servers. A server defines the abstract representation of its resource(s) and the operations on this representation. A resource can only be accessed or manipulated through its server. Because servers are constructed with well-defined interfaces, the implementation details of a resource are of concern only to its server. A server frequently acts as a client when it accesses resources managed by other servers, therefore, client and server are abstract concepts and do not necessarily imply differences in implementation.

All communication between processes takes the form of messages. RIG supports both synchronous and asynchronous message passing. A sender may or may not block waiting for the message to be queued at the destination, and he may or may not block awaiting a reply from the

receiver. A receiver may or may not block waiting for messages to arrive, and may receive messages asynchronously.

The OS consists of a collection of independent processes running on various computers and communicating through messages. Each process has an associated interface, whose job it is to define the functions of the process. The OS kernel provides the environment in which processes run. Typical kernel functions include message passing, process scheduling, and memory and device management. As well, each host in the network houses a number of server processes to handle OS system services such as file access and terminal communication.

Chapter IV

APPLICATION LAYER SERVICE ELEMENTS

The application layer is the highest layer in the OSI reference model providing the services and functions necessary for application processes in different open systems to communicate in order to perform a common distributed task.

Whereas each lower layer provides services to the next higher layer, the application layer provides services directly to the application processes.

An application process performs a defined set of functions established by a software designer and exists on one or more hosts and their associated terminals and peripherals. The functions performed by application processes may be implemented by software and/or hardware.

The resources required to support the application processes may be manual, automated, or industrial in nature. Human end users are manual resources, hardware and system software are automated resources, and sensors and actuaters are examples of industrial resources.

As well, application processes require access to local data bases (possibly shared by other application processes in the same or different systems) to store the results from the current application process, or to retrieve results from a previous application process.

Application processes may need to communicate with other application processes to accomplish tasks larger than those they could accomplish individually. That is, a single application process may be assigned a small part of a larger complex task. The OSI reference model only deals with that part of the application process which supports the OSI protocols needed for it to communicate with another application process located on any other open system.

That part of an application process participating in the open procedures is called the "application entity" and is the only part of an application process within the scope of the OSI reference model. In other words, the application entity is that part of the application process providing compatible communication and co-operation with other application processes via their application entities to accomplish common objectives. For example, parts of an application process may be hosted within the environment of a single system and may communicate among themselves using the internal interprocess communication facilities provided by that system. Such parts, together with their "closed" interconnection arrangements, are considered to be outside the scope of OSI.

As discussed in Chapter 2, each layer is composed of these peer entities which communicate with other peer entities in the same layer by executing the peer protocols. Figure 4.1 illustrates how a connection can be established between two application entities. Once the connection is made, the application protocols are used across the connection, allowing the application processes to communicate with each other.

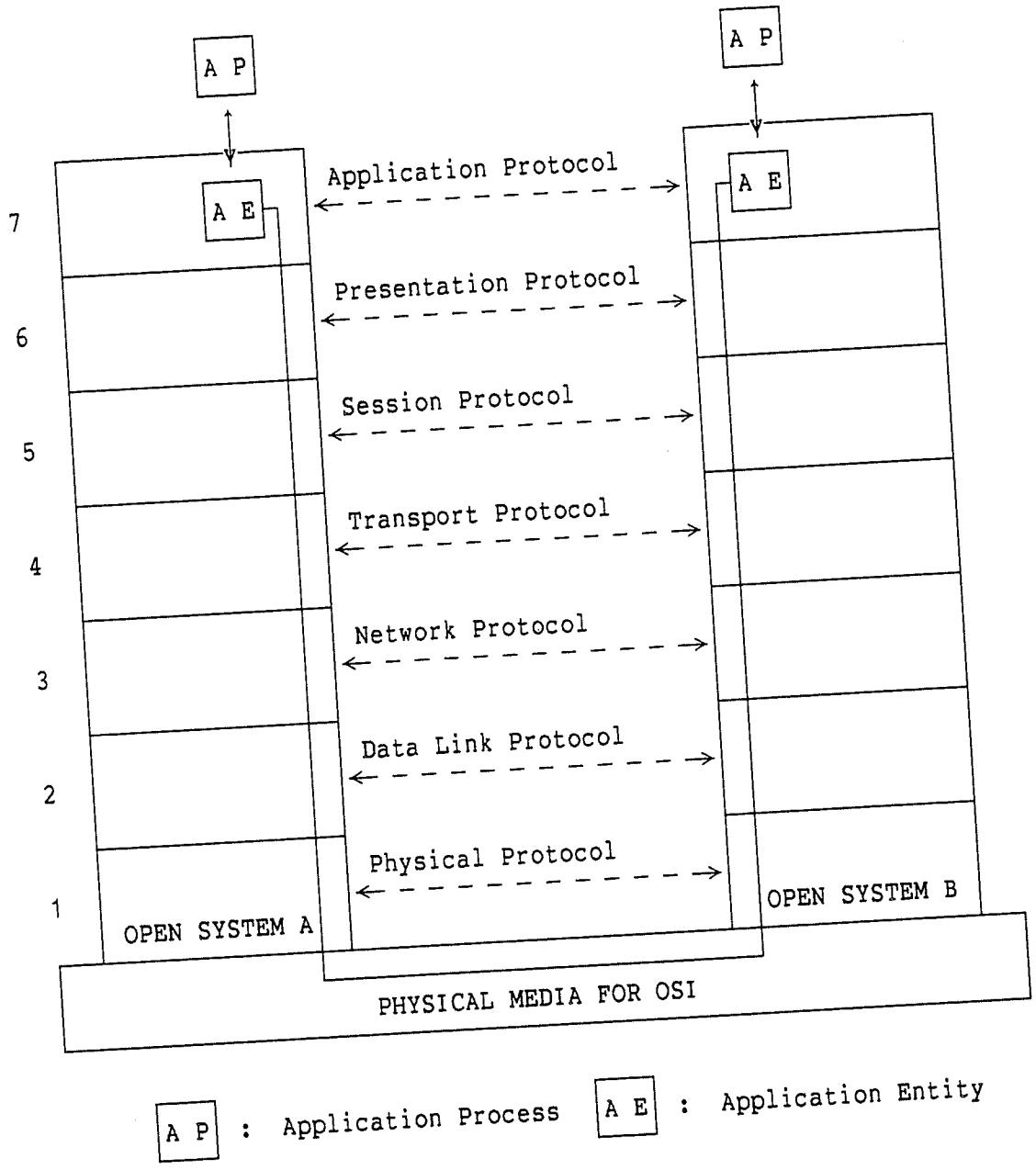


Figure 4.1: Protocol Layers of OSI

In each open system, the application layer consists of a set of identifiable service elements, each of which accepts and processes requests for provision of some OSI capability or provides to the application process some response as a result of a stimulus (a resource

for example) from the OSI environment [Barto83]. Any given subset of such service elements constitutes a unique type of application entity.

Figures 4.2 and 4.3 depict the application entity structure that is being considered by ISO. In this structure, three categories of service elements are recognized: application independent service elements (or common application service elements), system service elements (or application specific service elements), and user specific service elements. Each application entity may consist of service elements from all three categories. Note that Figure 4.3 does not imply any specific relationship between the service elements.

In the following sections we will describe the relevance of each of these service elements to the application layer. In the case where we cannot pinpoint their applicability to the application layer, we will use the term "higher-level".

For the sake of simplicity, we will not describe these service elements to the same depth or in the same manner as given in [Barto83]. For example, the application independent service element definition recommendation given by ISO, partitions the service elements into a number of capabilities such as origination, termination, context, interruption, information transfer, status, dialogue control, and synchronization. A number of parameters needed by each type of application independent service element are then defined. Our discussion of service elements will be limited to those capabilities needed by the application processes, and will be given at a higher level of abstraction than outlined by Bartoli. It is expected that during

1984 the service elements of the application layer will be completely defined and agreement will be reached on the application protocols.

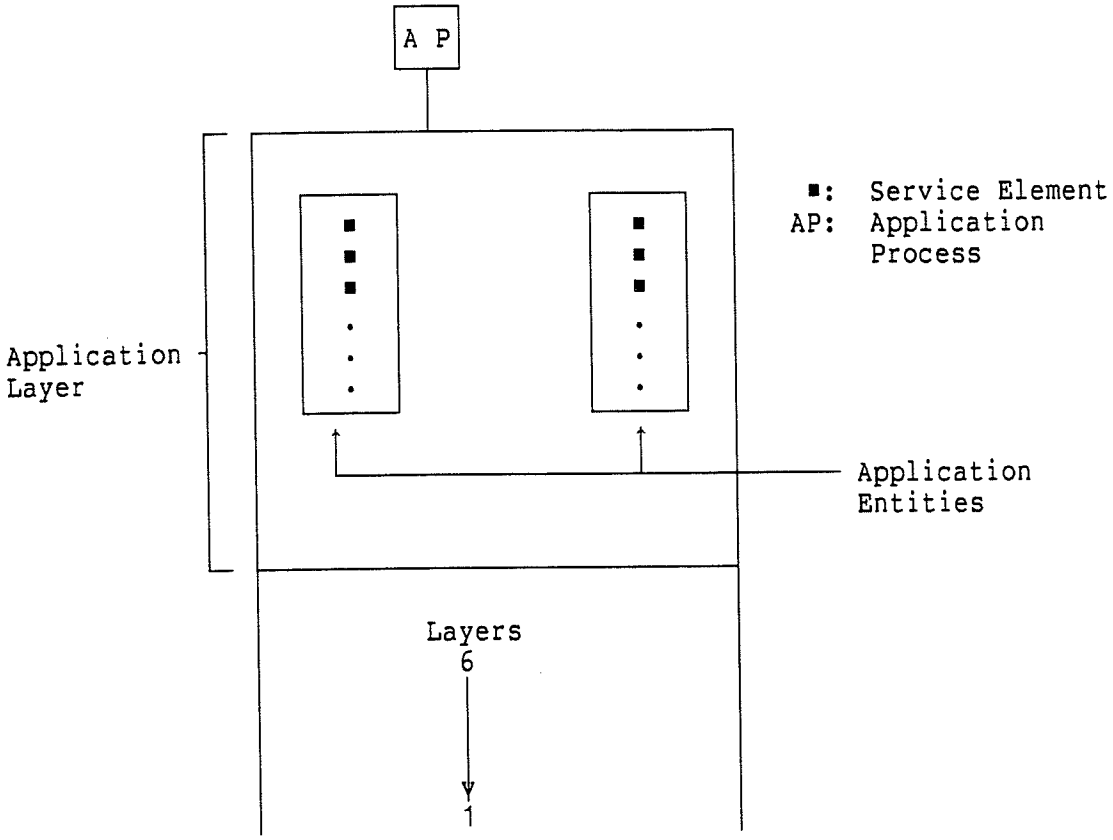


Figure 4.2: Processes, Entities, and Service Elements

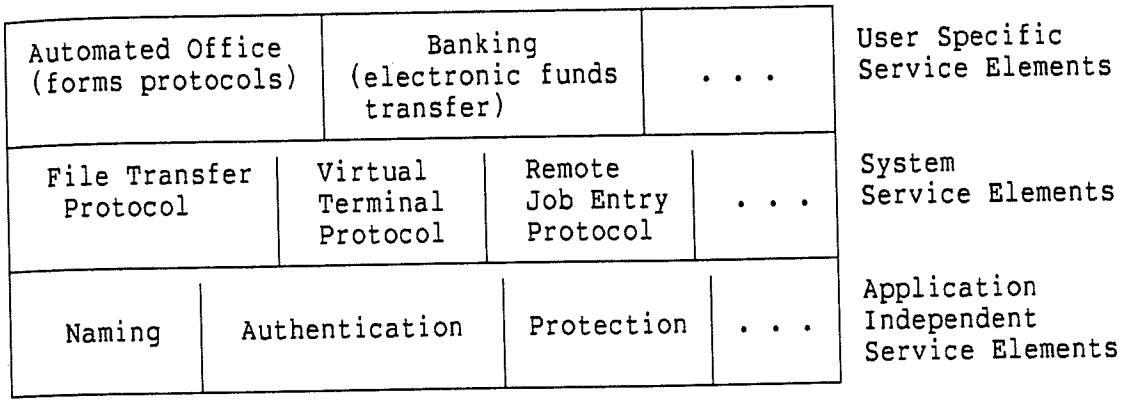


Figure 4.3: Structure of an Application Entity

4.1 APPLICATION INDEPENDENT SERVICE ELEMENTS

Application independent service elements provide capabilities required by the application processes for information transfer independent of the nature of the application. Setting up an association between application processes and terminating the connection are two such examples.

Users of interconnected open systems have a wide range of information processing applications which need the use of various resources for computing, data storage, data communication, as well as data input and output. Because certain resources may be duplicated within the interconnected systems, users may seek to gain increased resilience or availability by identifying backup resources to be selected in the event of failure. At the same time system managers may seek to retain control of the ways in which their systems will collaborate in meeting user needs.

To provide such services, open systems will need to communicate, among themselves, information relating to the management of the resources. In this way resource use may be planned, organized, supervised, and controlled (see Section 2.3.9 and the discussion of management protocols). In this section we wish to discuss five application independent service elements which could serve for the management of open systems: identifiers (naming), error control, resource management, synchronization, and protection.

4.1.1 Identifiers (Naming)

Identification systems (often called naming systems) are at the heart of all computer system design because it is only through the use of identifiers that access and manipulation of resources, processes, and other such mechanisms, can be accomplished.

An identifier is a string of symbols, usually bits or characters, used to designate or refer to an object [Lamps81]. We use the term object to refer to resources such as processors, storage, input/output devices, processes, files, directories, and the like.

There are three important classes of identifiers widely used in distributed systems: names, addresses, and routes. Shoch makes a useful informal distinction between these classes of identifiers: the name of a resource indicates what we seek, an address indicates where it is, and a route tells us how to get there [Lamps81]. Names used within the application layer and other such higher layers have often been called ports, or logical or generic addresses, distinct from physical or

specific addresses. If an identified object is to be manipulated or accessed, the identifier must be mapped using an appropriate mapping function and context into another identifier and ultimately into the object. Commonly, names are mapped into addresses and addresses into routes, although one could map a name to a route and a route to an address.

Higher-level naming schemes are required to meet the needs of human users for their own local mnemonic names, and to assist them in organizing, relating, and sharing objects. These high-level naming conventions can be embedded explicitly within particular applications or be supported by special name servers for example. Mapping from a high-level name to a machine-oriented identifier can be explicit through (high-level, machine-oriented) identifier pairs, or implicit by having some algorithmic or other relationship between the structures of the identifiers.

The concepts of addressing and routing are intimately related because routing is the level of identifier mapping that allows actual access to processes. The choice of address structure has a significant impact on how addresses are mapped into routes, on whether the mapping can be distributed, and on how easily the network can be extended and reconfigured.

A good illustration of the use of addressing and routing at the application layer is a naming-graph implemented with directories to create and access higher-level names. The leaves of the naming-graph are names of resources other than directories; the non-leaf nodes are

directories. The path name of a resource is the sequence of branches (directory entries) traversed from the root to a node. A given resource may be reachable by several paths, allowing users to share resources by specifying his own path name. Directories could be extended to maintain machine-oriented names for multiple copies of a resource. Application programs would be responsible for tracing down a naming-graph according to a path name to obtain the machine-oriented names at the leaves. This service could be provided in the form of a directory server for example.

4.1.2 Error Control

Error control is concerned with detecting and attempting to recover from errors and failures of different types. Error control mechanisms may be as simple as designing the form of state information so that duplication will not cause harm, or as complex as those used to assure consistency of update in distributed data base systems (data integrity).

In the extreme case, when the underlying services have failed and have been unable to recover, it remains for error recovery to be initiated by service elements at the application layer. An error is said to have occurred when part of the system is incorrect, therefore, the term "error recovery" is equivalent to the term "erroneous state recovery". The way error recovery is accomplished may depend greatly on how error detection and isolation are conducted. If it is possible to reduce the consequences of an error, the error recovery scheme becomes simpler than when the consequences cannot be identified. The enforcement of atomic operations (that is, when a fault occurs, leading to the failure of a system component, every currently executing

operation must be either completed or undone) in conjunction with error confinement mechanisms make error recovery easier to achieve.

As an example, consider two recovery schemes, backward error recovery (B-recovery) and forward error recovery (F-recovery). B-recovery techniques utilize recovery points and involve undoing everything a system has done since the last recovery point. A recovery point is a record of all needed information whereby a consistent state of (part of) the system may be established. F-recovery techniques depend on the possibility of detecting the causes and consequences of a fault and correcting them, and involve doing again only those things which have been detected as being incorrect.

4.1.3 Resource Management

There are a wide variety of resources being locally managed within the application layer and other higher layers, including buffer space, access to communication channels, CPU cycles, address spaces, and so on. Allocating and scheduling resources at the various distributed hosts and nodes are usually based on local decisions because of the need for local autonomy and because knowing how to allocate and schedule on a global basis is not well understood.

In distributed systems, one would like to use resource management protocols that could achieve both low delay and high throughput. Delay refers to the time interval between the time the process is ready to send a message until the first useful bit of the message reaches the destination. Throughput refers to the number of useful data bits per second that reach the receiver in some interval.

Delay is affected by the transmission and queuing properties of the IPC mechanisms, and by the number of overhead setup messages that may have to first be exchanged to reserve resources, authenticate the sender, etcetera, before the desired request or data can be sent. Throughput is affected by the former as well, and by the amount of overhead information that must accompany the sending or receiving messages. Therefore, mechanisms to achieve both low delay and high throughput may require long term retention of some state information and some preallocation of resources.

4.1.4 Synchronization

Synchronization refers to mechanisms used by co-operating (application) entities to co-ordinate access to shared resources or order events. The purpose of synchronization mechanisms is to provide processes with some means whereby the system may be kept in a consistent state. Example forms of synchronization are co-ordinating the initialization, maintenance, evolution, and termination of distributed shared state information used for protection, resource management, data translation, and error control; and ordering access by processes to shared information.

Typical synchronizing mechanisms are semaphores, hardware interlocks, and so forth. Semaphores are non-negative integer variables operated on by the synchronizing primitives P and V. If we let S be such a semaphore variable, then V(S) increases the value of S by 1 in a single indivisible action; the fetch, increment, and store cannot be interrupted, and S cannot be accessed by any other process during the

operation. The operation P(S) decreases the value of S by 1, if possible. If S=0, then it is not possible to decrement S further; the process invoking the P operation then WAITs until it is possible. The successful testing and decrementing of S is also an indivisible operation.

Typically such a semaphore resource is used by several processes to co-ordinate exclusive access to a shared resource (a file for example). In this case, after the semaphore value S is initialized to 1, each process performs a P(S) operation to lock the resource and await notification before accessing it. After accessing the resource, each process performs a V(S) operation to unlock the resource.

In a distributed system these mechanisms may be centralized or distributed, meaning either the mechanism possesses a unique name, known by all processes which must synchronize with each other, or there are multiple occurrences of the mechanism distributed throughout the system.

4.1.5 Protection

Whether hardware and software faults occur because of deliberate attempts of one user to maliciously interfere with another user or parts of the system, or because of accidental "bugs", control of access to network components (be they data, programs, computer power, storage space, or peripherals) is still a fundamental part of any system.

The protection needs of distributed and centralized systems are the same, namely to ensure that only authorized (application) entities can read, modify, or manipulate resources; that unauthorized entities cannot

prevent authorized entities from sending, modifying, or manipulating resources; and that only authorized entities can communicate.

Watson states that all security systems are built on some level of trust within a single host [Lamps81]. This means that either all processes trust one another, constitute a single protection domain, or all processes can only access each other and each others' resources through a well defined and defensible interface. If we assume the high-level operating system and access interfaces can be trusted, then encryption techniques can be used to protect the information in transit against errors, protect the subnet against wiretapping, establish safe authentication protocols (identifying each communicating partner to the others), detect deliberate message modification or playback, and allow secure passing of access control between processes.

4.2 SYSTEM SERVICE ELEMENTS

System service elements provide capabilities to handle the needs of specific information transfer processes, such as file and job transfers, and data base accesses.

We will provide an overview of three of the more important system service elements and the protocols they execute: virtual terminals, file transfer and access, and remote job entry.

4.2.1 Virtual Terminals

The most common computer communication application is terminal access [Green82]. Not only is there an enormous number of physical terminals on the market today, but all of them differ significantly in line length, meaning of attentions or "breaks", character sets, cursor addressing conventions, data rates, and so on. No application program exists that successfully will adapt to all of the hundreds of real terminal types available (indeed, this would be impractical if not impossible).

Terminal handling is more of a problem with networks than with a single host system. For example, if the network was composed of a number of homogeneous nodes, then terminal data could simply be "packaged" and transferred as is. Unfortunately, the system we have defined must handle every kind of terminal that may access it.

Most computer installations supporting a limited number of terminals, carry out terminal support through the use of software device drivers in the operating system. Each device driver handles a particular type of terminal attached to it. If this solution is taken for a network supporting N terminal types, then each of the M hosts on the network would have to support N terminal types, giving M x N terminal drivers. This is clearly impractical. The forthcoming protocols attempt to reduce this "M x N" problem to a manageable "M x 1" problem.

The most common solution is to define a network Virtual Terminal (VT). A virtual terminal represents the model of an "ideal terminal" equipped with enough adjustable parameters so that it can emulate a

large number of real terminals. The virtual terminal uses virtual terminal protocols (VTP), or rules for setting parameters and exchanging information with the VT. A virtual terminal protocol defines a set of procedures and message formats for the support and control of a wide variety of real terminals. The protocol exists between a pair of virtual terminals, using ports and names from the transport liaison.

Users of the VTP include ARPANET in the United States, CYCLADES network in France, GMD network in West Germany, and many vendors. The VTP approach is to compose input at a terminal, translate data to the standard VT format, and transfer the information to the receiving host which translates the data to a form expected by the host's terminal handling software. The VTP never needs to know the exact nature of any physical terminal because it addresses a set of functions implemented in a standard fashion. This allows terminal-application and terminal-terminal communication in a functional compatible form, irrespective of the type of physical terminal used [Davie79].

The first VTP was the ARPANET Telnet (Telecommunications Network) protocol. The success of Telnet grew out of three basic concepts: the network virtual terminal, negotiated options, and a symmetrical view of terminals and processes. To keep the protocol simple, it was only intended for scroll-mode terminals (one-dimensional type). From the work done on Telnet and by European investigations into VTPs, it was found that a well defined VT, and the development of a model for attentions or interrupts, are crucial in a general VTP.

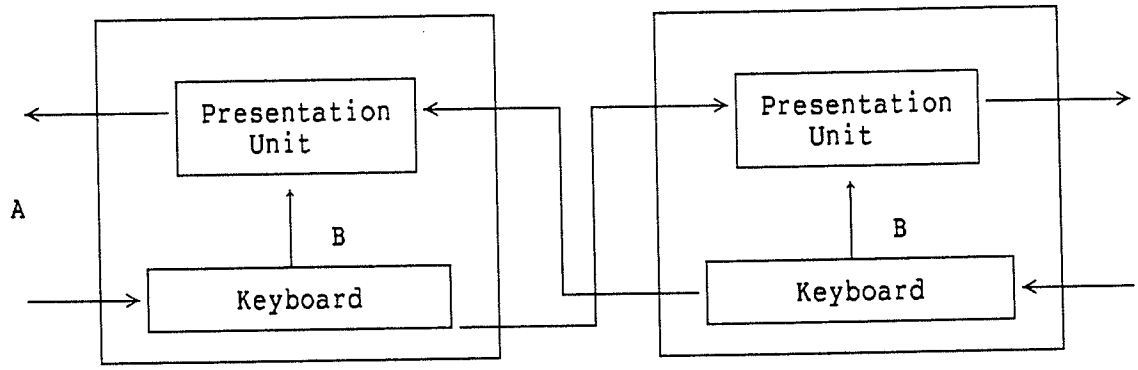
Most of the research done with VTPs has mapped them on top of an end-to-end transport service (for example, the host-host protocol in the ARPANET). This is equivalent to the upper three layers of the OSI reference model. The VTP has actually been split over several layer boundaries. For example, functions of creating and maintaining the VT are part of the presentation layer, while functions of dialogue control and data delimiting, if any, are a session layer service.

The virtual terminal protocol is initiated by a user at a terminal accessing an application program. In a heterogeneous network environment, it can be used to support terminal-terminal or application-application configurations. Record communication and teleconferencing are two examples of a terminal-terminal configuration. A less obvious configuration, application-application, is used to connect two programs originally designed for use by a terminal user. This scheme is used many times in the ARPANET [Green82].

Two models of VTP include the Telnet Model and the Asymmetrical Model. The Telnet Model, earliest of its kind, takes a symmetric view of terminals and processes (see Figure 4.4). The VTP connection exists as two VTs where the keyboard of one is connected to the display, or presentation unit, of the other, and vice versa. The representation of data on the local terminal or process is mapped onto its VT "keyboard", and from there it is sent to the presentation unit of the remote VT, which maps the data to its representation. That is, each VT has the ability to convert process or real terminal representations into a VT representation, and to convert a VT representation into a format needed by its associated host. In some applications, the displays shown by the

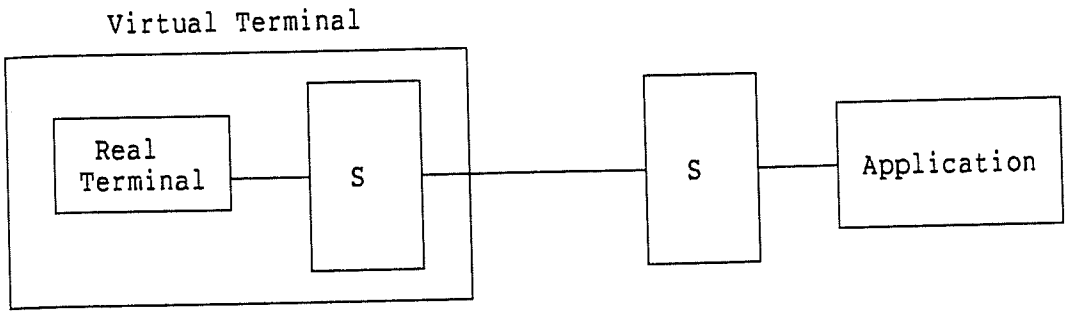
two VTs may differ for short or long periods of time. Echo, performed by none, one or both VTs, is controlled by a negotiation sequence.

With the Asymmetrical Model, a VT consists of the real terminal and the software needed to adapt the local representation to the standard language (that is, the VT format) (see Figure 4.5). The model is asymmetric; that is, it can only handle terminal-application configurations. At the receiving host there exists software to convert the VT format into the representation expected by that host. If the application needs the status of the VT for some reason, it must send a command to the VT and await a reply. To prevent subsequent input from the user from invalidating an undelivered reply, half-duplex operation is used. This helps control contention for the VT and, as a result, less asynchrony is allowed than with the Telnet model. No echo exists because terminal input is displayed locally.



A: Terminal or Application Process
B: Local or Remote Echo Capability

Figure 4.4: The Telnet Virtual Terminal Model



S: Adaption to the Standard Language

Figure 4.5: The Asymmetrical Virtual Terminal Model

The central component found in all virtual terminals is the data structure. The data structure is an abstract representation of data being "displayed" on the VT through the presentation unit, and consists of pages, lines, and character positions available for characters to fill. The most general form of the data structure displayed by the presentation unit is a two-dimensional array (referred to as the virtual terminal page), containing R rows and C columns. Associated with a page is a pointer to the current character position. In this case, the pointer would represent the vector (R,C). The page may be further subdivided into Z fields. A field is a contiguous set of positions sharing a common attribute. For example, the protected/unprotected attribute is classified as a field access control attribute for the sophisticated two-dimensional array [Bauwe78].

There are some marked differences between the data structure and the presentation unit, even though the data structure has been described by those attributes commonly associated with the description of presentation unit capabilities. For example, the presentation unit may

represent exactly one data structure (as in the case where the dimensions of the screen coincide with those selected for a page data structure); or part of a data structure (as with multi-page terminals which contain enough memory to handle several page data structures for display to the screen); or several successive data structures (a screen containing several line data structures) [Bauwe78].

Another difference between the data structure and the presentation unit is evident if a line data structure is longer than the physical line of the presentation unit. In this case, an operation called line folding displays the data as a sequence of two or more physical lines. This operation is local and therefore, invisible to the application.

In summary, a given application is only interested in the data belonging to the current data structure. That is, if the screen is displaying several line data structures, then the application is only interested in the current line. Similarly, the application is only interested in the unique logical co-ordinate indicating a particular element's position in the line data structure (rather than a physical co-ordinate described in terms of the presentation unit). The physical and logical co-ordinates are equivalent only if the dimensions of the data structure and presentation unit are identical. The concepts of logical co-ordinates and of data structure have end-to-end significance, whereas the concepts of physical co-ordinates and of presentation unit have local significance [Bauwe78].

Most VTP proposals contain the following six phases of operation taking place between a terminal and an application (or in general, between two processes) [Green82]:

1. Establishment Phase - this phase is entered on a request of VT service until VTP connection is established.
2. Negotiation Phase - since terminals provide a variety of functions, a VTP includes a negotiation mechanism allowing both partners to agree on the VT characteristics which they are going to use. This phase is entered when one side or the other wishes to change parameters or classes, or use new terminal primitives.
3. Form Definition Phase - this phase is entered for data-entry (two-dimensional) terminals only, and is used to define a new form on the terminal's presentation unit.
4. Data Phase - entered for the actual exchange of data between two partners.
5. Synchronous Attention Phase - entered to handle attention or out-of-band signals.
6. Termination Phase - this phase is entered when the VT service is requested to terminate VT service on this connection.

Other phases may be included, depending on the particular classes of terminals used.

The use of attentions (phase 5 above) are of concern to VTP designers. An attention condition provides a "break" function found in many terminals used to recover from an uncertain or abnormal condition. Asynchronous attentions are independent of the data stream and pose no problems. Synchronous attentions do pose problems because their actions must be co-ordinated with a point in the data stream. Issues to be resolved at time of attention include whether or not to purge the data, the dialogue mode after the attention (full or half duplex), and, if

half duplex, which side begins. Both Green and Bauwens describe an attention scheme developed by Duenki and Schicker in which the above attention information is carried by the attention telegram; a control signal sent by the attention initiator to establish dialogue possibilities, purge information, and so on [Bauwe78], [Green82].

During the data phase, dialogue control, or the synchronization of alternate dialogue, must be established. In full duplex or free running mode it is up to the protocol and its users to sort out what is happening. Data are moving in two directions at the same time. The operator and application may want to access the presentation unit at the same time, therefore, a locally defined priority rule must be set up to settle such conflict. While one or the other partner is being denied access to the presentation unit, the data the unit is generating must be stored in the virtual terminal until the unit is freed. In half duplex or alternate mode, data travels between partners, but not simultaneously. The presentation unit is allocated alternately to the partner that has the right to speak. While an operator is waiting for his turn, he may still enter data (buffered at the VT) until he receives his turn, at which time transmission of the typed-ahead data will proceed.

4.2.2 File Transfer and Access

The most important requirement in a distributed network of computers, after the ability to access computer resources from interactive terminals, is the necessity to handle files in a standard way. A capability is needed to allow transfer and access of files between heterogeneous systems in the distributed environment.

One common file transfer solution is to define a network-wide file management system (NFMS), allowing file transfer and access between heterogeneous systems in the network.

A NFMS contains two components: the service points to provide access to the local file handling facilities, and the control points to monitor and synchronize the operation of service points so actions can be performed on files, as requested by users or application programs (see Figure 4.6 (a)). A transaction is composed of one control point and one or two service points. To complete the transaction, conversation takes place between the control process (controller) at a control point and the server process (server) at the service point. A file transfer involves two servers and a controller. One is the producer server (source file) and one is the consumer server (destination file).

The tasks of the file transfer controller are twofold:

- Control Phase - identifies the site where the files reside and determines whether the file is to be transferred to a new file or appended to the existing one. As well, any access keys providing verification of authorization and access rights must also be specified. Negotiation of file attributes (code used, data compression) is also performed during this phase.
- Data Phase - begins with its control phase between source and destination servers to ensure correct protocol operation. This phase is responsible for recovery in the event the transfer fails.

Both controller and server dialogue take place through the transport service, locally provided by a transport station residing on each site

(that is, an interprocess communication mechanism establishing liaisons between processes' ports).

The above protocol involves three liaisons: two between the control point and the source and destination servers, and one between source and destination servers. An alternative method to this is to place the control point with the entire source or destination server, and use one communication channel (liaison) to multiplex both control and data information (see Figure 4.6 (b)). This minimal architecture for file transfer simplifies synchronization and restart problems between both ends [Gien78].

Recovery is an important function. The protocol must be able to restart the operation at the point of failure. This becomes more complicated in a network environment because of the non-compatible record structures of the various systems.

A checkpointing mechanism is needed to keep track of the data transfer. One typical method is as follows. During the data transfer phase, the source sends checkpointing commands in the data stream. The commands are numbered sequentially 1 through N; when command M is received successfully, checkpoint M-1 can be discarded. Associated with any given command M, is a marker with the position in the data stream occupied by checkpoint M. The destination sends a reverse acknowledgement to the source indicating successful receipt of the marker. To ensure that the checkpoint information itself is not lost, all such information is stored on disk or tape. When failure occurs, the file transfer can be restarted from the marker associated with the last checkpoint.

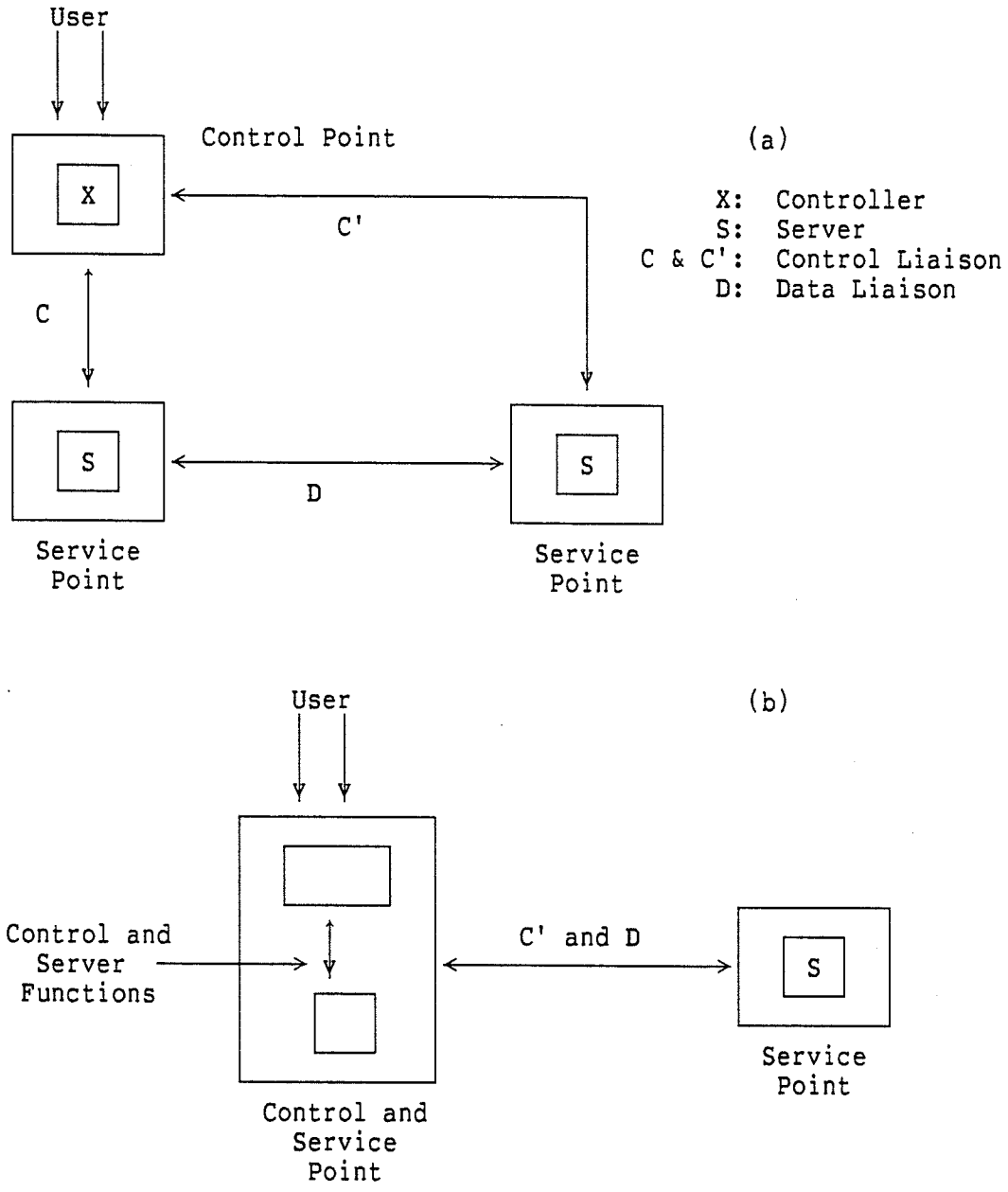


Figure 4.6: Two File Transfer Architectures

4.2.3 Remote Job Entry

The file transfer protocol enables users to transfer files across the network. Some of these file transfers will be associated with the

submission of jobs. The user will require a standard network facility to define such a job. In the most general case this will involve the transfer of files between a variety of systems in a well-defined sequence with actions defined on these files. A good example is the transfer of PL/1 source code, from a local to remote system, for compilation and execution. The data for the execution are obtained from a second remote file. The user initiates all this from a third system, expecting a log of status information about the job, while output from the job is transmitted to a fourth system for printing and plotting. The total operation involves three file transfers, and is controlled by a job entry protocol at the system on which the job is executed. The user communicates with the job entry protocol from his terminal using a virtual terminal protocol.

Before achieving a remote job entry protocol of the generality described above, a simplier subset is required which will allow the definition of a complete job as a single file, transport it for execution and return the results to a location specified by the user.

As well as scheduling the series of events associated with job execution, the remote job entry protocol must inform the user of job progress and completion, and restart file transfers if failure occurs in any of the systems.

4.3 USER SPECIFIC SERVICE ELEMENTS

User specific service elements provide capabilities to specific application processes (such as office automation and order entry) and are not subject to recommendation by ISO.

Thus, user specific service elements are the user contributions to the application entities. For example, consider automated office systems (AOSs) as a typical application. Service element capabilities that would be specific to this application are a suitable programming language design to support the clerical user and the parallelism within the office, and the specification of electronic mail protocols to support a number of generic commands to retrieve information; compose and edit messages; request message delivery, forwarding, and replies; and so forth. We will discuss these through the use of specific examples in the next chapter.

Chapter V

INTEGRATED DISTRIBUTED SYSTEMS

There are two trends particularly evident in the transition from networks to distributed systems. First, the complexity and sophistication of hardware, software, procedures, and protocols have increased. This is because we are trying to unify a number of separate heterogeneous computers instead of simply interconnecting them in some ad hoc fashion. At the same time, the proliferation of computer systems of both small, medium, and large scale, and the dramatic change in the relative cost of computing versus communications, has given the user a myriad of aids for information storage, retrieval, communication, manipulation, and control. This too increases the overall complexity of the system. Second, the level of integration has increased. However, this integration has not progressed at the same rate as the conglomeration of automated aids and applications sweeping the distributed environment. Integration, the most important unifying concept of all, allowing a high degree of interaction and cross fertilization, and enabling the different areas and technologies to mesh together, has been given less emphasis.

Integration is formally defined as the combination of parts to create a whole [Sykes76]. This implies some form of synergy in which the system components are tied together into a whole that is greater than the sum of its parts. The value added is directly proportional to the

degree to which the different functions of these components are integrated into a total system.

While distributed systems have made a significant impact on the integration problem, they still lack the synergy necessary for total unified control and access. If we extrapolate and improve upon the potentials of a distributed system, we get an integrated distributed system.

An Integrated Distributed System (IDS) is a distributed system designed in a co-ordinated fashion so that the separate components employ commonality of data fields, data structures, programs, and protocols, wherever this can improve the effectiveness or future evolution of the system.

Integration here takes place on at least two fronts: functional integration and system integration. Functional integration refers to the need for the user's model of a system to be complete and consistent. The user must be able to work in an environment that provides all of the facilities he or she will need in order to perform the work without having to learn several different command languages or subsystem models. System integration refers to the need for the operating systems, programming languages, architecture, and data bases that converge into a single, uniform environment. We will refer to the successful combination of functional and system integration in a distributed system as distributed integration.

Table 5.1 outlines the most important issues discussed in this thesis relating to the integration of distributed systems. The important point

to remember in the integration process is that objects (be they processes, resources, servers, human users or whatever) must be able to communicate with each other coherently, transparently, and in a generalized way which organizes the environment into a consistent homogeneous structure. Each of the issues discussed directly or indirectly contribute to this process. In this chapter we will revisit each of these issues in an attempt to exemplify their importance to distributed integration.

TABLE 5.1

Issues for Achieving Distributed Integration

- OSI reference model
- Communication subnet characteristics
- Distributed data
- Distributed computation
- Distributed control
  - High-level operating systems
  - Interprocess communication
- Application layer service elements
  - Identification
  - Error control
  - Resource management
  - Synchronization
  - Protection
  - Virtual terminals, file transfer
  - User specific service elements

5.1 THE OSI REFERENCE MODEL

The purpose of OSI is to allow any computer anywhere in the world to communicate with any other, as long as both obey the OSI standards. The approach adopted by ISO and subcommittee 16 was to use a layered architecture to break up the problem into manageable pieces. The OSI reference model is a framework for co-ordinating the development of OSI standards. In OSI, the problem is approached in a top-down fashion, starting with a description at a high level of abstraction which imposes few constraints, and proceeding to more and more refined descriptions with tighter and tighter constraints. We found this framework to be an appropriate foundation for the development of integrated distributed systems because it is the only internationally accepted framework, has a good technical base, is well thought out, and is gaining wide acceptance.

For example, the Xerox Network System (XNS), a distributed office information system, adopts an enhanced OSI reference model with seven layers [Dalal82]. Protocol layers permit different kinds of office services to be added as the need arises, thereby allowing an organization to minimize the initial purchase cost, control any system expansion, and assist in integrating the miscellaneous elements into a powerful system.

It is expected that other groups developing standards for other areas or large corporate systems will benefit from the OSI reference model and use it to organize and co-ordinate their work in communications, data bases, and integrated distributed systems.

5.2 COMMUNICATION SUBNET CHARACTERISTICS

The issue of subnet internal operation surfaced in Chapter 2 with a discussion of datagrams and virtual circuits. A brief summary reveals that virtual circuits require a destination address only during setup, error handling (that is, solving the problems caused by damaged, lost, or duplicate packets) is transparent to the host (that is, done in the subnet), end-to-end flow control is provided by the subnet, messages are always passed to the host in the order sent, and initial setup is required. Datagrams on the other hand require a destination address in every packet, error handling is explicitly done by the host, end-to-end flow control is not provided by the subnet, messages are passed to the host in the order they arrive, and initial setup is neither required nor possible. One concept not yet mentioned is that the choice between virtual circuit and datagram as an interface and as an internal strategy are independent. For example, DECNET supports a datagram service implemented by using datagrams inside the subnet. Error control, sequencing, and flow control are all handled by the transport layer. SNA, on the other hand, uses virtual circuits inside the subnet to provide virtual circuit service to the transport layer. ARPANET uses a mixture of virtual circuits and datagrams; the hosts see the subnet as a virtual circuit service, but the subnet actually uses datagrams internally.

Even though datagrams and virtual circuits are not a direct concern at the application layer, they directly affect the range of layered high-level protocols providing additional service to the application layer and ultimately to the user. Many proponents of a datagram

interface ([Lamps81], [Lantz82], [Liskov79], [Mimic82]) believe that communication in a DOS consists mainly of control messages between parties that simply send a message and receive a reply. Datagram based networks were designed to deal with these bursty communication patterns. Connection or virtual circuit protocols are still necessary however, in order to provide many useful services to a completely integrated system, such as error handling, flow control, sequencing, bulk data transfer, and so on.

A combination of both services is provided in XNS. At the network layer, a datagram service is used. This service does not provide reliable delivery of datagrams; it simply gives its "best efforts" to deliver each one, and allows the end processes to build protocols which provide reliable communication of the quality they themselves desire. To allow the sender and receiver to engage in additional conversation (such as for a file transfer operation), a virtual circuit service is implemented at the transport layer. This allows all packets to be orderly and reliably transported between sender and receiver. Furthermore, it removes several limitations inherent in the lower level datagram service. For example, it is not constrained by a finite packet size (datagrams in XNS are limited to 512 bytes) and can transport a stream of data, rather than just packets.

5.3 DISTRIBUTING DATA

Distributing data contributes to the overall reliability and efficiency of the integrated distributed system. In many cases there is a natural distribution of data. Each company plant maintains the data relevant to its personnel, inventory, orders, and production. Most accesses are then to local data, but occasionally one location needs data from one or more other locations. Thus, if parts of the data are distributed to areas that use the particular data frequently, then there will be faster access to this local data, improved data protection, and improved control.

In other cases there is no natural distribution, but the user may want to keep important data at two or more sites for increased reliability. It is not hard to imagine, for example, that a bank whose head office is located in an earthquake zone might want to keep a duplicate of its account data base in a distant city, even if keeping both copies identical is complicated and expensive. In some applications (for example, military ones), even short down times due to hardware failures are completely intolerable, hence the need for replication.

As well, distributing data contributes to the maintainability of the integrated distributed environment. For example, the works manager of NSW provides a distributed system data base for maintaining global state information such as directories of process names, global file catalogues, user authorization and accounting information, tool descriptors, and so on.

5.4 DISTRIBUTING COMPUTATION

Distributing computation is another important issue we spent some time discussing in Chapter 3. We analyzed two ways in which a single user problem can be divided into pieces and run on different machines: the hierarchical model and the horizontal model.

As an example, consider the user-server model of horizontal distributed computing. In this model, each user has a personal minicomputer with perhaps some local disk storage. Depending on the application, the personal computer may range from a 16-bit machine with 64 kilobytes of memory and a floppy disk, to a 32-bit machine with several megabytes of memory and a 100 megabyte hard disk. Systems of this type are discussed in [Abrah80] and [Clark80].

In the user-server model, all personal computers are connected by a high speed LAN, allowing users to share data and expensive peripherals, and send each other mail. The user does all his actual computing on his personal computer, but various server machines exist to carry out specific functions on behalf of any user who requests them. For example, the system could have several disk servers, whose function is to read and write raw disk blocks, without regard to their contents. Typical messages that could be sent to a disk server are requests to allocate, return, read, and write disk blocks.

Since raw disk blocks are too low a level of abstraction for most applications, the system would probably have one or more file servers, offering file system services. Typical messages that can be sent to a file server are requests to open, close, read, and write files. The

file servers are responsible for maintaining directories and connection status information to turn the primitive disk services into a usable file service.

Different file servers may offer completely different file systems. One file server may regard a file as a linear sequence of bytes, with operations to read and write contiguous portions of a file, and little more. Another file server may regard a file as a linear sequence of variable length records, with operations to insert and delete records, as well as reading and writing them.

The system would also provide data base servers for those users who require a more efficient and powerful access to data. A data base server might accept requests about the data base, analyze them itself, and return the answers. The data base servers could use either the file servers or the disk servers, as appropriate.

Thus, a resource can only be accessed or manipulated through its server. Because servers are constructed with well-defined interfaces, the implementation details of a resource are of concern only to its server.

Distributing computations using models similar to the above provides the parallelism, asynchrony, and reliability needed by the application processes to control an ordered set of tasks working toward a common goal.

Access to resources is network transparent. An application process can access a local resource with the same semantics it uses to access a

remote resource. The result is an environment in which application processes can communicate with servers without regard for the topology of the distributed system as a whole.

5.5 DISTRIBUTING CONTROL

We found that distributing control can be accomplished through the use of a high-level operating system. In Chapter 3 we described two common structures: an operating system implemented and used as the base native operating system on a single system of common architecture, and one implemented as a guest layer. Initially, many HLOSs will likely be implemented as guest systems, on top of existing OSs, but over time, as part of the evolution toward distributing computing, it is expected that the structure of base OS design will evolve toward that required for a distributed OS.

The only way system integration can be achieved is for all processes, resources, and users to be able to communicate with each other coherently and in a consistent manner. For example, consider the user-server model of distributed computation. For any given application, it will be necessary for the user to access many servers (resources) and for the servers to exchange information in order to accomplish a task. This is an essential goal of distributed integration, one which can only be attained through IPC. The IPC mechanism we are about to present emphasizes the underlying processes needed to co-ordinate the user-server machines discussed in the previous section.

Britton and Stickel discuss a type of IPC facility called the requestor-server model (RSM), which provides a conceptual framework appropriate for structuring distributed applications and information flow (in a user-server environment for example) [Britt80]. The basic unit in the system is the process, and applications consist of a group of co-operating concurrent processes. Two types of processes exist in this environment: the servers and the requestors.

The servers provide a service or resource. A server may require the service of another process, therefore, it can exist as both a server and requestor. Processes exchange information (requests) through the use of messages. The distributed computing environment consists of a number of shared resources which are controlled by the server processes. The requestors make service requests, one at a time, to a server process for a specific resource, and it is up to the server to provide mutual exclusive access to the resource. The identity of the requestor remains unknown to the server. In this particular configuration, each server has a "mailbox" from which it receives the requests. The server must either respond to the request, make sure some other server responds (that is, it can delegate the workload as seen fit), or queue the request for later execution in the event some server already has the resource.

A requestor is a process which uses the resources and services provided by a server. This process is analogous to calling a subroutine: call the server for some service, and get a response of completion with possible returned values. Each call operates asynchronously; that is, it does not have to wait for a response from a

server before it continues. Of course, if the requestor process needs the resource immediately in order to continue, then it will have to WAIT. By using sequence numbers in the requests to keep track of responses, the requestor can issue requests to more than one server concurrently.

5.6 APPLICATION LAYER SERVICE ELEMENTS

This section covers again those service elements introduced in the previous chapter we find most appropriate to distributed integration. The application independent service elements (identification, error control, resource management, synchronization, and protection) are those underlying mechanisms needed for the management of the distributed environment, and represent the system integration component of distributed integration. Many of these service elements are more difficult to handle in a distributed system. Since system integration can only be achieved through resolution of these problems, we will discuss the problems they pose and possible solutions to them.

System service elements (virtual terminals, file transfer, and so on) and user specific service elements represent the functional integration component of distributed integration. We will examine some of the problems the system service elements present in a distributed environment and then illustrate by a number of examples, the role system service elements and user specific service elements play in achieving functional integration.

5.6.1 Identification

The problems introduced to the design of identification systems by distribution result from the heterogeneous nature of host systems, each with their own possible local identification system for memory addressing, files, processes, and other objects; and problems introduced in maintaining distributed contexts and mapping information, compounded by delays and errors in message transmission and local system or network crashes.

The specific questions that need to be addressed are what type of identifiers are required at the application layer and other higher layers of the distributed system, and how should we deal with heterogeneity.

We can begin to answer these questions by reviewing some of the more important identification system goals. First, it is generally agreed upon that at least two levels of identifiers, one convenient for people and one convenient for machines, be supported. There should be a clean separation of mechanism for these two levels of identifiers. A machine-oriented identifier should be a bit pattern easily manipulated and stored by machines, with protection, resource management, and other mechanisms. A human-oriented identifier, on the other hand, is generally a human readable character string. The human-oriented name service could be embedded in application processes or be provided by general purpose name servers using data base management or directory techniques.

Second, it may be appropriate to provide a system viewed as a global space of identified objects rather than one viewed as a space of identified host computers containing locally identified objects. Similarly, the boundaries between physical components and their connection as a network should be invisible, to the extent possible. (This does not mean that users cannot discover or influence an object's location.)

Third, the relocation of objects should be supported. This implies there should be at least two levels of identifiers, a name and an address, and that the binding between them be dynamic (that is, our ability to relocate, extend, or share objects can be achieved if names do not have to be bound to addresses or routes until they are used).

5.6.2 Error Control

Developing error control mechanisms in general is a complicated task for a number of reasons. For example, redundant information needed for error control must be correctly initialized and maintained in the face of arbitrary message delays, message errors, and system crashes. Additionally, information needed for error control itself must be transmitted across a hazardous environment, where it can be misaddressed, lost, damaged, duplicated, or missequenced.

In general, perfect error control (detection and correction of all errors) cannot be achieved at the application layer, or any lower layer for that matter, because the above two problems cannot be completely solved simultaneously within a given layer. All that can be done is to

decrease unrecoverable error frequency by increasing the probability of error detection and recovery. Error control mechanisms at a given layer must be developed around a knowledge of the type of errors that can occur, their frequency, and the structure of the service being protected.

As an illustration, consider a virtual terminal protocol at the application layer. The VTP assumes that a transport protocol at the lower level guarantees an arbitrarily low probability of misaddressed, damaged, and missequenced information. However, some errors cannot be recovered in case of major system failure. Therefore, service elements must provide for checkpoints; that is, the possibility to backtrack up to a point in the past from which activities may be restarted safely. Taking a checkpoint involves some context saving. This must be done at each end of the conversation when the two communicating processes are in synchronized states.

5.6.3 Resource Management

Developing mechanisms to use resource management philosophies that can achieve both low delay and high throughput is a difficult task. Tradeoffs exist between quantity and duration of retention of state information, amount of overhead information carried with messages, and the number of messages needed to initialize state information. For example, if the user wants high throughput but only wants to retain state information and allocate resources during a conversation, he can use initial exchanges of messages to set up the state and resources required during the conversation. The result may be high delay caused by these setup messages.

The important conclusion is that mechanisms to achieve both low delay and high throughput will require relatively long term retention of some state information and some preallocation of resources [Lamps81]. The goal is to find techniques that minimize the state and preallocated resources required for naming, error control, protection, etcetera, while keeping enough to achieve the desired delay-throughput balance.

5.6.4 Synchronization

Entities involved in communications must remember a certain number of parameters, or state variables. As a whole, this information is termed the protocol context. It is occasionally necessary to make sure that both sides of a context are simultaneously in a well-defined state (for example, at initialization, resetting checkpointing, closing, and so on). This is what we defined as synchronization in the previous chapter. Synchronization is intended to guarantee context or state consistency, or else to report an error.

The challenge in distributed systems is that there is no ideal observer capable of freezing each entity in a well-defined state. In a nondistributed system, using shared memory, all co-operating entities "see" the same state. This is not the case in a distributed system. Even without errors or node failures, it will be impossible for each entity in a co-operating set to maintain the same view of their total state as the others, due to arbitrary delays in message updating or in reporting state.

Only received messages can give clues about the state of the conversation as it was at some point in the past. When error control context is not yet set up or has been destroyed, messages exchanged are not protected against loss or duplication. Therefore, the building of error control context must work even in an adverse environment.

5.6.5 Protection

Distributed systems, by their nature, imply some amount of physical access from one host to the data of another. A secure system must provide software that protects against remote violations of privacy. It is not absolutely clear just what software designs will ensure privacy and security in distributed systems and be less subject to subversion than software in nondistributed systems. The protection problem is compounded, of course, in systems consisting of heterogeneous hosts and nodes, physically distributed components, and multiple controlling authorities.

To describe a protection system we must describe the terminology which it is using. Consider the terms cipher, encipherment, encryption algorithm, cleartext, ciphertext, and key. A cipher is an algorithmic transformation performed on a symbol-by-symbol basis on any data. Encipherment or encryption is the application of a cipher to data. An encryption algorithm is any algorithm implementing a cipher. Cleartext or plaintext is the input to the encryption algorithm, and ciphertext is the output from the algorithm. A key is used to control the transformation performed on cleartext to encipher it. Conventional ciphers use the same key to encrypt and decrypt a message. Public-key

ciphers, on the other hand, use a pair of keys representing the inverse of each other, such that neither is deriveable from the other.

The most widely used cipher in the United States for encryption is the Data Encryption Standard (DES). It was designed at IBM and approved in 1976 as a standard by the National Bureau of Standards. The DES enciphers a 64-bit message block under control of a 56-bit key to produce a 64-bit ciphertext. The same key is used for both enciphering and deciphering plaintext.

One mode of the DES, called the cipherblock chaining mode, is used at higher levels of architecture where character length independence can be assumed [Nelso79]. This mode operates only on 64-bit data blocks using a feedback technique to encipher and decipher the blocks. Rather than treating the individual blocks of a message independently, they are connected together during encipherment. This is done to avoid the danger common with the Electronic Codebook Mode (ECB) that for any given key K, the same 64-bit input will always produce the same 64-bit output.

A good reason for the use of block chaining is given in [Lamps81]. In this example, the message is a money transfer transaction. Only one or two blocks out of this transaction are used in specifying the amount of money involved. As long as the intruder knows the format of these blocks, he can increase or decrease the money value (even though he knows nothing about the actual contents of the blocks). This form of subversion assumes the blocks have been enciphered independently. Other modes of the DES are discussed in [Needh78], [Nelso79], [Raslt83] and [Voydo83].

Nelson states that the higher the level in the architecture at which the crypto functions are performed, the more restricted and secure can access to sensitive information be made [Nelso79]. Functions can be implemented and performed virtually at all levels in the OSI reference model, but a more reliable and practical approach must restrict the functions to some small number of levels.

Consider cryptosystems at the application level. This level of security already exists in many systems and will continue to exist as the need for high levels of protection in communication and file management prevails. Systems at this level are usually totally implemented in software, although some data encryption hardware peripherals may be used as well. The application programmer is responsible for key management and control of the crypting process. This produces high overhead on individual applications, and could lead to disaster if the key is lost through malfunction or misuse (file contents may become lost and possibly irretrievable). Application level cryptosystems usually employ the cipherblock chaining mode of the DES for efficiency and generality for communications and other protocols. File management systems can take advantage of the hardware and software cryptosystem elements being used at this level as well.

5.6.6 Virtual Terminals

In a nondistributed system, all users usually have the necessary knowledge about which real terminals are available in the system and what their parameters are (screen dimensions for example), therefore, virtual terminals are not required. Users would use the real terminals

according to this knowledge and the operating system should prevent their misuse (and inform the users by error messages).

In a computer network, virtual terminals and their protocols are devised to hide the differences between different kinds of terminals. For example, it would be frustrating for the user to try to log onto a remote computer only to be told that the program he wants to use does not talk to his brand of terminal. By selecting the appropriate virtual terminal, the user can avoid this undesirable situation.

In a distributed system, the same properties are desired. However, in a distributed system, the virtual terminal selection procedure is handled automatically. As a consequence, distributed systems need to achieve agreement by negotiation (at the beginning of an application connection for its whole duration or dynamically, whenever this is required) what particular virtual terminals the communicating partners have at their hosts. These means are part of the system service elements.

5.6.7 File Transfer

In a distributed network environment, the user needs the ability to access and handle files in a standard way. Ideally, the file transfer should not be affected by the data's location in the distributed system. In reality, a number of location dependent factors affect file transfer. For example, faster access can be achieved when the application program and data are located in the same computer system. As well, storage costs and quality of security vary from system to system. A typical

file transfer protocol includes rules to open and close sending and receiving files for data transfer and error recovery, and for data translation through a user supplied exit or protocol option.

Copying a file involves taking the entire file and writing it on a given destination device. The copy must be named and if source and destination devices are not compatible, some restructuring (for example, file block and record) may be performed. As well, some systems use their own file compression techniques to save storage, while others represent literal information differently. Heterogeneity of file names, structure, access procedures, and data representations, contribute to the overall complexity of file transfer and management.

5.6.8 User Specific Service Elements

To illustrate the importance of system service elements and user specific service elements, we would like to take the reader through three examples. The first example illustrates the use of VTPs to manipulate and manage tools and processes in the office. The second example illustrates the use of a high-level distributed language in an office environment to achieve integration of office tasks. The third example discusses a number of mechanisms to integrate the user interface.

MANIPULATING OFFICE TOOLS

A relatively new approach to the manipulation and management of tools and processes in an office (particularly those systems based on a model

of distributed computation, such as RIG) allows the user to engage in multiple, concurrent activities, all from the comfort of his "multiple window" display device. The basic concept implemented in the RIG system allows the user to view the output of various application programs (which includes facilities such as text processing, decision support, and time management) on different areas of his screen. Many similar systems allow the user to create windows on the display, move windows around, change their size, and overlap windows ([Begbi83], [Lantz79], [Meyro82]).

A window manager based on virtual terminal protocols is needed for the management of screen space. Many of these systems are meant to simulate pieces of paper on a desk and to give the user more than one logical line of communication. For example, the user can read electronic mail in one window, edit the file referenced in the mail in a second window, and compile that file using a third window.

The virtual terminal management system (VTMS) supported by RIG converts the device-dependent characteristics of a single physical terminal into multiple device-independent virtual terminals [Lantz79]. In RIG, each activity initiated by the user is handled by a VT. A VT is roughly equivalent to an independent physical display device, therefore, each VT may accept output simultaneously (although only one VT at a time may be used for user input). Let us look at the components of RIG's VT and then see how such a VTMS is implemented.

A RIG virtual terminal has three logical components: line, pad, and window. A line is the VT's current source of input. Processes share

input and output, but while output can be multiplexed in time and space, input can only be entered using one window at a time (in other words, there exists only one "active" line at a time). The input line is generated from characters leaving the user's keyboard, or by typing a special pre-defined function key. The characters are buffered to allow "type-ahead". This means that the characters will be accepted as fast as the user can type, but will not appear on the active line until they are extracted from a queue in response to a request for input. Input from any VT may be processed a character at a time, a page at a time (for editing files), or a line at a time (for user commands).

A pad is a disk-based data structure used to store and edit VT output. The data structure is two-dimensional and cursor addressable, indexed by a (row,column) pointer. Each line may have a number of attributes (reverse video, blinking). To handle recovery from system crashes, the pad stores a fixed number of lines in memory and uses two scratch files for temporary disk storage. In this way, most output can be salvaged in the event of a crash. The VTMS supports a number of full-screen editing features available to processes through the pads (insert line, delete line, splitting lines, string location and substitution, and so on). The pad may be mapped onto multiple VTs, allowing two or more users to "link" or share screens. Attention processing is supported to suspend or discard output to a pad.

A window represents the mapping of a VT onto a display device. The VTMS supports the decomposition of screen space into windows, viewports, regions, and images, to give the user freedom over the logical arrangement of information on the screen, to assist application

processes with the use of several VTs, and to configure windows on the display by size, contrast and content.

To co-ordinate the aforementioned office activities, each RIG terminal has its own terminal input and output handlers. A screen handler is created initially to handle windows, and a line handler is created initially to co-ordinate all user input. Processes exist to handle pads, and all the controlling processes comprise the VT controller (VTC) which manages all VTs associated with the user. A virtual terminal protocol is encorporated to administrate communication between application processes and the VTC.

OFFICE PROCEDURES BY EXAMPLE (OBE)

Office procedures cannot be integrated without a suitable programming language design to support the clerical user and the parallelism within the office.

Office procedures are dynamic - they affect the global environment. Concepts such as compile time, load time, and run time are really of no interest to the user. A system allowing the user to represent his requirements in symbolic form and to "run" the program is most appropriate.

Query-By-Example (QBE) is a two-dimensional programming language allowing users to describe their application directly to the computer. The system is used by non-programmers to retrieve, modify, define, and control existing data base information. An extended version of QBE used

specifically for office functions is called Office Procedures by Example (OBE). While QBE is an IBM product released in 1978, OBE is a research project [Zloof81].

OBE is a superset of QBE; developed not only to handle simple relational data base queries, but to perform word processing, data processing, applications development, electronic mail, forms and report, and graphics, all using one unified language. The user programs use two-dimensional pictures of business objects (tables, letters, reports, graphs, and so on). Many objects may be processed simultaneously, but not all objects may be displayed at the same time. Two-dimensional programming is more than inputting numbers on a spread sheet; it means entering expressions anywhere on the object and having OBE's two-dimensional parser extract and interpret these expressions.

At the core of OBE is its data base system allowing easy extraction of data, and mapping such data into the bodies of the various objects. Another key concept of this system is its attempt at integrating the objects. Objects are not passive; they can be gifted with a certain amount of intelligence. For example, OBE allows users to express various trigger conditions. Once the condition is activated, one or more actions take place without user intervention. The user for example, could send messages at a predetermined date or frequency. Acknowledgements, creation of logs, inventory replenishments, all could be performed automatically. Combining triggers in the user's applications allow complete automation of complicated office procedures. Objects can be passed around, resources can be used, and information can flow freely and intelligently between components of the office system, all through the controlling OBE program.

THE USER INTERFACE

The user's primary concern in a distributed integrated environment is to get a job done, as simply as possible. He has little or no interest in the peculiarities of the different systems to be used, such as the syntax and semantics of the various command languages. At best, the command language provided should allow maximum access to the features of all the systems available with minimum system-dependent interaction [Lantz80].

One of the first attempts at a coherent and consistent user interface was Officetalk-Zero, a first generation information system that became operational in 1977. The system was developed by members of the Office Research Group at Xerox Palo Alto Research Center (PARC), and implemented in an environment of minicomputers connected by high speed communication networks.

The hardware consists of a number of Xerox Alto minicomputers, disks, and advanced display units. A "mouse" pointing device is used for controlling the cursor on the screen. A mouse is a small box, held in the palm of the hand, and moved about across any surface. The mouse's underside contains wheels or a ball which corresponds to movements of the cursor on the screen. The top of the mouse has a button, which when depressed, records the x-y co-ordinate of the cursor.

The Officetalk system contains a number of work stations. The fundamental data object used in the system is the form, and replaces

paper at the work station. Single page forms and files of forms are electronically passed among the work stations.

The user interface is similar to RIG's: each work station provides a graphical window onto a worker's desk; forms are manipulated using a pointing device (mouse) and a keyboard. Officetalk attempts to combine useful subsystems like a text editor, graphics package, communication facility, filing facility, and forms data entry facility, into a single integrated system.

Officetalk is a distributed program executing on at least one minicomputer. A second minicomputer is used as a file server, maintaining a data base of electronic mail transactions, profile tables for user authentication, or tailored blank forms to be used in a given application. Each user has his own local minicomputer, or work station, for performing work. Most of the user's information state is kept in the file server rather than at each station.

Officetalk emphasizes the integrated user interface and the concept of modeless environments. A modeless environment attempts to prevent locking the user into a specialized and highly restricted state while in the mode. The editing commands are always applicable and other functions (such as copying and messaging) are at hand as well. At any point in the editing session, the user may switch to another function without first terminating the current mode. The concepts of windows and view ports are supported, so that windows may be reduced, enlarged, moved, scrolled, and overlapped at will.

The developers of the Xerox Star went one step further with the "desk-top" conceptual model and implemented icons: concrete objects represented by pictures on the display screen. Physical objects, such as paper folders, file cabinets, and mail boxes, have as their electronic counterparts, pictures on the high resolution display device [Meyro82]. For example, to open a file and read it, the user would simply point the mouse at the picture of the appropriate paper folder.

5.7 THE COMPLETE SYSTEM

The issues discussed contribute to either functional integration or system integration, or in some cases to both. Table 5.2 shows the two components of distributed integration and lists those issues most

TABLE 5.2

Distributed Integration Components

FUNCTIONAL INTEGRATION
- OSI reference model
- System service elements
- User specific service elements

SYSTEM INTEGRATION
- OSI reference model
- Communication subnet characteristics
- Distributing data, computation, and control
- Application independent service elements

important to each component. Using this table as a guideline, let us look at some of the system integration and functional integration components of a hypothetical integrated distributed system.

5.7.1 System Integration

System integration can be achieved using a uniform user-server interaction model. Single users work on powerful personal "work stations" and share specialized "servers". The work stations are matched to the needs of different workers: professionals, secretaries, word processing operators, and others. The shared servers provide services such as file storage, authentication, communication to remote hosts and other distributed systems, data base access, and so on.

The underlying mechanism to handle the distributed computations is the requestor-server model of IPC. Processes send messages and wait for the arrival of messages. If no application process is waiting for them when they are sent, the messages are queued. Synchronization is assured by associating the occurrence of a given event with a message. Access is indirect: a given process acts as a server handling the resource on behalf of other processes.

The HLOS consists of a kernel on each host and provides the execution environment for processes. Functions provided by the kernel include basic message passing, process scheduling, memory management, and device management facilities. Each host supports its own complement of server processes, which might include a process manager for the creation, management, and deletion of processes; a job manager for the creation, management, and deletion of distributed jobs; a name server to maintain bindings from symbolic names for services to server addresses; and network servers to handle communication with other remote hosts, allowing applications to be written in a network-transparent manner.

Thus, the HLOS and its IPC facility provide uniform access to all servers and their controlling processes, whether they are local to the user or remote from him. This is a necessary prerequisite for system integration.

A name lookup and resource location service is needed to provide a mapping between the name and the address of a process within the system. A name server can be used to handle this function. The name server may have access to a distributed data base to perform name lookup and resource location. Its directory may be decentralized, distributed, and replicated; the system would most likely be optimized for efficient lookup and less efficient in its update operations (the entries in the directory are expected to change relatively infrequently, when compared with the frequency of lookups). A data base server answers queries from application processes and communicates with other data base servers in order to maintain the distributed directory.

Name service is used only at the initiation of communication. Thereafter, the specific address of each process is known to the other. This type of name service is an example of dynamic binding at first use - the time an application process first requests a resource.

An integral part of the IDS will be a specialized server whose function is to authenticate the originator of a request. The technique that could be used is encryption, based on keys distributed by the authentication server [Clark80]. The authentication server, when invoked by the originator of a request, sends a unique key to each end of the potential communication, along with information that specifies

the identity of each end. When the request originator then uses this key to communicate with the other end, the fact that each can decode the messages of the other assures each that the other is what it claims to be.

Other aspects of the user-server interaction would include error recovery and resource management.

5.7.2 Functional Integration

The work station provides the user with uniform access to all available facilities through a single display terminal. A user sitting at his work station can view the output of various application programs on different areas of his screen. As described in Section 5.6.8, system service elements could provide the capabilities for such an integrated configuration. Built on top of this is a high-level distributed language (such as OBE) to co-ordinate all user activity and allow new services to be constructed from existing ones.

For example, consider the following business project scenario. The user sits down at his work station and enters all current information into his data base. This information might include sales or invoicing information as it occurs, and each time a transaction (such as a sale) occurs, the new information is entered and accumulated. A business manager then uses this information to generate sales and commission reports. Selected information can be summarized by salesmen and then used by a program to project future period sales. Past, present, and future information is compiled, summarized, and used for decisions.

Graphics can be generated on screen or paper to make past, present, and future comparisons easier. Summarized information, graphs, and projections can then be included in end-of-period reports provided by a word-processing program. The word processor inserts these figures and graphs into a template report and surrounds them with text. From project start to finish, user intervention is not required. And because this project is done at regular intervals, the system just repeats the report's last procedure to generate the next one. This is a good overview of a useful application for an integrated distributed system.

This application would use an integrated work station; a number of high-level distributed languages to initially set up the project logic; and distributed servers to handle access to all relevant data required, and to co-ordinate information exchange among the various peripherals (printers, graphics screens) and program packages (text editors, compilers).

We have demonstrated a system that uses proven mechanisms to achieve distributed integration (at least in theory). We are confident that more ambitious systems, if attempted with a similar emphasis on application level issues and the protocols they execute, would also prove to be successful.

Chapter VI

CONCLUSIONS

We began our discussion with an overview of computer networks, aiming directly at the application layer and its protocols and issues. As soon as we began to see what abstract issues must be addressed for the development of distributed integration, we moved our discussion toward distributed systems and application layer service elements.

We know that distributed integration must take place on at least two levels: at the functional (user) level and at the system level. At the functional level, the user's view of the system should be complete and consistent, providing uniform access to all distributed system facilities. Similarly, at the system level, the operating systems, programming languages, architecture, and data bases must converge into a single uniform environment. In other words, effective distributed integration service elements and protocols must be developed internally at the DOS and IPC level, and at the language level in order to accomplish these goals.

We found that the OSI reference model framework represents an appropriate foundation for the development of distributed integration because it is the only internationally accepted framework, has a good technical base, is well thought out, and is gaining wide acceptance. We also found that service elements and protocols discussed in the context of OSI, such as virtual terminal and file transfer, are an integral part

of the distributed integration design process. Through the use of such application layer service elements and protocols, an otherwise heterogeneous system can be organized into a coherent and consistent homogeneous structure.

In this thesis we have articulated a number of problems that must be solved in order for the integrated distributed system to be successful in the modern computerized world. The key we feel lies in the advancement of distributed systems with emphasis on designing and refining high-level service elements and protocols in general. Future research in the area of distributed integration will probably fall into two distinct but interrelated categories. The first category concerns the familiar technical problems associated with distributed system design: DOS structure, IPC mechanisms, and so on. The second category will use the theoretical and empirical results obtained from distributed systems to build the proper high-level service elements and protocols necessary to further extend and integrate the distributed system's capabilities.

REFERENCES

[Abrah80] Abraham, S.M., and Dalal, Y.K., "Techniques for Decentralized Management of Distributed Systems," **Compcon**, pp. 430-437, Spring 1980.

[Abram80] Abrams, M., et. al. (Eds.), Computer Networks: A Tutorial, IEEE Cat. No. EHO 162-8, 1980.

[Almes82] Almes, G.T., "Objects in the Eden System," **Electro/82 Conf. Record**, pp. 32-3/1-7, 1982.

[Ball79] Ball, J.E., et. al., "Perspectives on Message-Based Distributed Computing," **Comput. Networking Symp.**, IEEE, pp. 46-51, 1979.

[Barto83] Bartoli, P.D., "The Application Layer of the Reference Model of Open Systems Interconnection," **Proc. of the IEEE** 71,12 (Dec. 1983), pp. 1404-1407.

[Bauwe78] Bauwens, E., and Magnee, F., "The Virtual Terminal Approach in the Belgian University Network," **Computer Networks** 2,4/5 (Sept./Oct. 1978), pp. 297-311.

[Begbi83] Begbie, R., et. al., "iNet: The Intelligent Network," Bell-Northern Research and the Computer Commun. Group at the TransCanada Telephone System, 1983.

[Berns78] Bernstein, P.A., et. al., Tutorial: Distributed Data Base Management, IEEE Cat. No. EHO 141-2 (1978).

[Birre81] Birrell, A.D., et. al., "Grapevine: An Exercise in Distributed Computing," **Eighth Symp. Oper. Syst. Principles**, Dec 1981, in **Commun. of the ACM** 25,4 (Apr. 1982), pp. 260-274.

[Birre84] Birrell, A.D., and Nelson, B.J., "Implementing Remote Procedure Calls," **ACM Trans. Comput. Syst.** 2,1 (Feb. 1984), pp. 39-59.

[Black80] Blackshaw, R.E., and Cunningham, I.M., "Evolution of Open Systems Interconnection," **ICCC**, (Oct. 1980), pp. 417-422.

[Booth76] Booth, G.M., "Distributed Information Systems," **National Computer Conference (NCC)**, pp. 789-794, 1976. Also in [Liebo81, pp. 4-9].

[Britt80] Britton, D.E., and Stickel, M.E., "An Interprocess Communication Facility for Distributed Applications," **Compcon**, pp. 590-595, Fall 1980.

[Brown83] Brown, M.J., "The Complete Information-Management System," **BYTE** 8,12 (Dec. 1983), pp. 199-207.

[Canad82] Government of Canada, an Information Technology Publication (GES/NGI-20/G), A Guide to Open Systems Interconnection, Aug. 1982.

[Carde79] Cardenas, A.F., Data Base Management Systems, Allyn and Bacon, Inc., Boston, (1979).

[CCITT82] for period 1981-1984, question 27/VII, CCITT COM VII No. R9(C)-E, (May 1982), pp. 99-103.

[CCITT83] for period 1981-1984, question 5/VII, CCITT COM VIII No. R19(B)-E, (February 1983), pp. 1-177.

[Champ77] Champine, G.A., "Six Approaches to Distributed Data Bases," **Datamation** 23,5 (May 1977), pp. 69-72.

[Chand79] Chandy, K.M., and Misra, J., "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," **IEEE Trans. Softw. Eng.**, SE-5,5 (Sept. 1979), pp. 440-452.

[Chang83] Chang, D., "An Introduction to Integrated Software," **BYTE** 8,12 (Dec. 1983), pp. 103-108.

[Chora80] Chorafas, D.N., Data Communications for Distributed Information Systems, Petrocelli Books, New York, (1980).

[Chora84] Chorafas, D.N., Designing and Implementing Local Area Networks, McGraw-Hill, New York, (1984).

[Clark80] Clark, D.D., and Svobodova, L., "Design of Distributed Systems Supporting Local Autonomy," **Compcon**, pp. 438-444, Spring 1980.

[Cunni82] Cunningham, I. et. al., "Emerging Protocols for Global Message Exchange," **Compcon**, pp. 153-161, Fall 1982.

[Cypse78] Cypser, R.J., Communications Architecture for Distributed Systems, Addison-Wesley, Massachusetts, (1978).

[Dalal82] Dalal, Y.K., "Use of Multiple Networks in the Xerox Network System," **IEEE Computer** 15,10 (Oct. 1982), pp. 82-92.

[Davie79] Davies, D.W., et. al., Computer Networks and their Protocols, Wiley and Sons, New York, (1979).

[DEC76] Digital Equipment Corporation, "DECNET," (1976). Also in [Liebo81, pp. 293-315].

[DEC82] Digital Equipment Corporation, Introduction to Local Area Networks, (1982).

[Deuts82] Deutsch, D.P., "International Standardization of Message Transfer Protocols: An Overview," **Compcon**, pp. 162-167, Fall 1982.

[Donne79] Donnelley, J.E., "Components of a Network Operating System," **Computer Networks** 3,6 (Dec. 1979), pp. 389-400. Also in [Thurb81, pp. 253-273].

[Donov72] Donovan, J.J., Systems Programming, McGraw-Hill, New York, (1972).

[Ellis77] Ellis, C.A., "Consistency and Correctness of Duplicate Database Systems," **Oper. Syst. Review** 11,5 (Nov. 1977), pp. 67-84. Also in [Thurb80, pp. 3-36].

[Ellis80] Ellis, C.A., and Nutt, G.J., "Office Information Systems and Computer Science," **ACM Computing Surveys** 12,1 (Mar. 1980), pp. 27-60.

[Enslo78] Enslow, P.H. Jr., "What is a 'Distributed' Data Processing System?," **IEEE Computer** 11,1 (Jan. 1978), pp. 13-21. Also in [Thurb81, pp. 8-16].

[Envoy83] Electronic Messaging Service, Envoy 100 User's Guide, Draft No. 4, 1983.

[Falk77] Falk, G., and McQuillan, J.M., "Alternatives for Data Network Architectures," **IEEE Computer** 10,11 (Nov. 1977), pp. 22-31. Also in [Abram80, pp. 2/63-70].

[Faro81] Faro, A., et. al., "A Network Operating System for Local Computer Networks," **Proc. Real Time Systems Symp.**, (Dec. 1981), pp. 13-21.

[Forsd78] Forsdick, H.C., et. al., "Operating Systems for Computer Networks," **IEEE Computer** 11,1 (Jan. 1978), pp. 48-57. Also in [Liebo81, pp. 201-210].

[Garci80] Garcia-Molina, H., "Reliability Issues for Completely Replicated Distributed Databases," **Compcon**, pp. 442-449, Fall 1980.

[Garci82] Garcia-Luna-Aceves, J.J., and Kuo, F.F., "A Hierarchical Architecture for Computer-Based Message Systems," **IEEE Trans. Commun.** COM-30,1 (Jan. 1982), pp. 37-45.

[Gien78] Gien, M., "A File Transfer Protocol (FTP)," **Computer Networks** 2,4/5 (Sept./Oct. 1978), pp. 312-319.

[Green80] Green, P.E., "An Introduction to Network Architectures and Protocols," **IEEE Trans. Commun.**, COM-28,4 (Apr. 1980), pp.413-424.

[Green82] Green, P.E. Jr. (Ed.), Computer Network Architectures and Protocols, Plenum Press, New York, (1982).

[Green77] Greene, W., Major, "A Review of Classification Schemes for Computer Communication Networks," **IEEE Computer** 10,11 (Nov. 1977), pp. 12-21.
Also in [Abram80, pp. 2/53-62].

[Grieb80] Grieb, G.P., "Open Systems Interconnection - Management and Application Layer Protocol Issues," **ICC'80** (June 1980), pp. 5.3/1-4.

[Grime77] Grimes, J.D., "Distributed Processing Concepts Using Microprocessors," **Compcon**, pp. 140-144, Spring 1977.

[Gylys76] Gylys, V.B., and Edwards, J.A., "Optimal Partitioning of Workload for Distributed Systems," **Compcon**, pp. 353-357, Fall 1976.

[Horde78] Hordeski, M., Illustrated Dictionary of Microcomputer Terminology, Tab Books, PA, (1978).

[Infom83] Infomart, Videotex Application Layer Protocol Discussion Paper, June 27, 1983.

[Jense76] Jensen, E.D., and Boebert, W.E., "Partitioning and Assignment of Distributed Processing Software," **Compcon**, pp. 348-352, Fall 1976.

[Lamps80] Lampson, B.W., and Redell, D.D., "Experience with Processes and Monitors in Mesa," **Commun. of the ACM** 23,2 (Feb. 1980), pp. 105-117.

[Lamps81] Lampson, B.W., et. al. (Eds.), Distributed Systems - Architecture and Implementation, Springer-Verlag, New York, (1981).

[Lantz79] Lantz, K.A., and Rashid, R.F., "Virtual Terminal Management in a Multiple Process Environment," **Proc. Seventh Symp. on Oper. Syst. Prin.**, ACM, pp. 86-97, 1979.

[Lantz80] Lantz, K.A., "Command Interaction in Distributed Systems," **Compcon**, pp. 25-32, Fall 1980.

[Lantz82] Lantz, K.A., et. al., "Rochester's Intelligent Gateway," **IEEE Computer** 15,10 (Oct. 1982), pp. 54-68.

[Lauer79] Lauer, H.C., and Needham, R.M., "On the Duality of Operating System Structures," **Operating Systems: Theory and Practice, Proc. Intern. Symp.**, 1979, pp. 384.

[Lazow81] Lazowska, E.D., et. al., "The Architecture of the Eden System," **Oper. Syst. Review** 15,5 (Dec. 1981), pp. 148-159.

[Liebo81] Leibowitz, B.H., and Carson, J.H. (Eds.), Tutorial: Distributed Processing, IEEE Cat. No. EH0176-8, 1981.

[Linds84] Lindsay, B.G., et.al, "Computation and Communication in R*: A Distributed Database Manager," **ACM Tran. Comput. Syst.** 2,1 (Feb. 1984), pp. 24-38.

[Lisko79] Liskov, B., "Primitives for Distributed Computing," **Proc. Seventh Symp. on Oper. Syst. Prin.**, ACM, pp. 33-42, 1979.

[Locho83] Lochovsky, F.H., "A Knowledge-Based Approach to Supporting Office Work," **Database Eng.** 6,3 (Sept. 1983), pp. 43-51.

[Lorin80] Lorin, H., Aspects of Distributed Computer Systems, John Wiley and Sons, New York, (1980).

[Macfa83] Macfarlane, D., "What You Get When You Buy Office Automation," **Datamation** 29,2 (Feb. 1983), pp. 102-114.

[Marti81a] Martin, J., Computer Networks and Distributed Processing, Prentice-Hall, New Jersey, (1981).

[Marti81b] Martin, J., Design and Strategy for Distributed Data Processing, Prentice-Hall, New Jersey, (1981).

[Marya80] Maryanski, F.J., "Backend Database Systems," **ACM Computing Surveys** 12,1 (Mar. 1980), pp. 1-25.

[Meyro82] Meyrowitz, N., and van Dam, A., "Interactive Editing Systems: Part II," **ACM Computing Surveys** 14,3 (Sept. 1982), pp. 353-415.

[Mimic82] Mimica, O., and Marsden, B., "A Datagram-based Network Architecture for Microcomputers," **Comput. Commun. (GB)** 5,3 (June 1982), pp. 128-139.

[Mitch81] Mitchell, J.G., and Dion, J., "A Comparison of Two Network-Based File Servers," **Eighth Symp. Oper. Syst. Principles**, (Dec. 1981), in **Commun. of the ACM** 25,4 (Apr. 1982), pp. 233-245.

[Mokho79] Mokhoff, N., "Office Automation: A Challenge," **IEEE Spectrum** 16,10 (Oct. 1979), pp. 66-69.
Also in [Thurb81, pp. 611-614].

[Molli80] Mollis, L.L., "Open Systems Interconnection Upper Layers Activity," **Compcon**, pp. 558-563, Fall 1980.

[Moult80] Moulton, J., "High Level Protocol Boundaries in the ISO Model," **Proc. Trends and Appl.**, IEEE (May 1980), pp. 54-58.

[MTS81] Presentation given by Lorne Duke, Communications Specialist, Honeywell on "Network Architectures - Open Systems Interconnection".

[MTS82a] Memorandum from F.R.Kaita, General Staff Engineer, Subject: Synopsis of Local Networking Conference, Minneapolis, Minn., Nov. 2-6, 1981.

[MTS82b] (Annex II) Proposed Draft Recommendation - "Elements of a Network Independent Transport Service," 1982.

[MTS82c] (Annex I) Proposed Draft Recommendation - "General Explanation of the Reference Model for Public Data Network Applications," 1982.

[Needh78] Needham, R.M., and Schroeder, M.D., "Using Encryption for Authentication in Large Networks of Computers," **Commun. of the ACM** 21,12 (Dec. 1978), pp. 993-999.

[Nelso79] Nelson, J., "Implementation of Encryption in an 'Open Systems' Architecture," **Comp. Net. Symp.**, 1979, pp. 198-205. Also in [Thurb81, pp. 551-558].

[Olson82] Olson, M.H., and Lucas, H.C. Jr., "The Impact of Office Automation on the Organization: Some Implications for Research and Practice," **Commun. of the ACM** 25,11 (Nov. 1982), pp. 838-847.

[Ouste80] Ousterhout, J.K., et. al., "Medusa: An Experiment in Distributed Operating System Structure," **Commun. of the ACM** 23,2 (Feb. 1980), pp. 92-105.

[Pooch83] Pooch, U.W., et. al., Telecommunications and Networking, Little, Brown, and Company, Boston, (1983).

[Poulo80] Poulos, J.C., and Beavers, A.N., "Network Operating Systems: A Concept and Protocol," **Proc. Trends and Appl.**, IEEE (May 1980), pp. 64-69.

[Pouzi78] Pouzin, L., and Zimmermann, H., "A Tutorial on Protocols," **Proc. of the IEEE**, Nov. 1978, pp. 1346-1370. Also in [Thurb81, pp. 412-436].

[Ralst83] Ralston, A., and Reilly, E.D. Jr. (Eds.), Encyclopedia of Computer Science and Engineering, Van Nostrand Reinhold Company Inc., New York, (1983).

[Reid80] Reid, B.K., "Addressing and Delivery Issues in Computer Mail Systems," **Compcon**, pp. 161-164, Spring 1980.

[Renss79] Rensselaer, C., "Centralize? Decentralize? Distribute?," **Datamation** 25,4 (Apr. 1979), pp. 88-97.

[Rober78] Roberts, L.G., "The Evolution of Packet Switching," **Proc. of the IEEE** 66,11 (Nov. 1978), pp. 1307-1313. Also in [Abram80, pp. 2/46-52].

[Salte83a] Salter, M., "Melding the Office of the Future into a Compatible Whole," **The Financial Post, Special Report - The Office**, November 5, 1983, p. S1.

[Salte83b] Salter, M., "Integration: The Final Goal of Automation," **The Financial Post, Special Report - The Office**, November 5, 1983, p. S3.

[Solom79] Solomon, M.H., and Finkel, R.A., "The Roscoe Distributed Operating System," **Proc. Seventh Symp. on Oper. Syst. Prin.**, ACM, pp. 108-114, 1979.

[Sprou78] Sproull, R.F., and Cohen, D., "High-Level Protocols," **Proc. of the IEEE** 66,11 (Nov. 1978), pp. 1371-1386.

[Steel80] Steel, T.B. Jr., "CCITT Perspectives on Higher Level Protocols," **ICC**, pp. 5.5/1-4, 1980.

[Stone78] Stone, H.S., and Bokhari, S.H., "Control of Distributed Processes," **IEEE Computer** 11,7 (July 1978), pp. 97-106.

[Stone79] Stonebraker, M., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES," **IEEE Trans. Softw. Eng.**, SE-5,3 (May 1979), pp. 188-194.

[Strud81] Strudwick, P.A., and Adkins, R.E., "Integrating Voice and Data at the Human Interface: The Key to Effective Communication in the Office of the Eighties," **NTC**, pp. D1.2/1-3, Fall 1981.

[Sykes76] Sykes, J.B. (Ed.), The Concise Oxford Dictionary of Current English, Oxford University Press, Oxford, Sixth Edition, (1976).

[Tanen81a] Tanenbaum, A.S., "Network Protocols," **ACM Computing Surveys** 13,4 (Dec. 1981), pp. 453-489.

[Tanen81b] Tanenbaum, A.S., Computer Networks, Prentice-Hall, New Jersey, (1981).

[Thorn80] Thornton, J.E., "Backend Network Approaches," **IEEE Computer** 13,2 (Feb. 1980), pp. 10-17.

[Thurb80] Thurber, K.J. (Ed.), Tutorial: Office Automation Systems, IEEE Cat. No. EHO 172-7, 1980.

[Thurb81] Thurber, K.J. (Ed.), Tutorial: A Pragmatic View of Distributed Processing Systems, IEEE Cat. No. EHO-160-2, 1981.

[Toda80] Toda, I., "DCNA Higher Level Protocols," **IEEE Trans. Commun.**, COM-28,4 (Apr. 1980), pp. 575-584.

[Tsich80] Tsichritzis, D.C., and Lochovsky, F.H., "Office Information Systems: Challenge for the 80's," **Proc. of the IEEE** 68,9 (Sept. 1980), pp. 1054-1059.

[Uhlig81] Uhlig, R.P. (Ed.), Computer Message Systems, **Proc. IFIP TC-6 Intern. Symp. Comput. Message Syst.**, April 1981.

[Voydo83] Voydock, V.L., and Kent, S.T., "Security Mechanisms in High-Level Network Protocols," **ACM Computing Surveys** 15,2 (June 1983), pp. 135-171.

[Watso80] Watson, R.W., "Network Architecture Design for Back-end Storage Networks," **IEEE Computer** 13,2 (Feb. 1980), pp. 32-48.

[Wecke80] Wecker, S., "DNA: The Digital Network Architecture," **IEEE Trans. Commun.**, COM-28,4 (Apr. 1980), pp. 510-526.

[Wu80] Wu, S.B., and Liu, M.T., "Assignment of Tasks and Resources for Distributed Processing," **Compcon**, pp. 655-662, Fall 1980.

[Yakub83] Yakubaitis, E.A., Network Architectures for Distributed Computing, Allerton Press, New York, (1983).

[Zimme76] Zimmermann, H., "High Level Protocols Standardization: Technical and Political Issues," **Proc. Third Inter. Conf. Comput. Commun.**, (Aug. 1976), pp. 373-376.

[Zimme78] Zimmermann, H., and Naffah, N., "On Open Systems Architecture," **Proc. Fourth ICCC**, pp. 669-674, 1978.

[Zimme80] Zimmermann, H., "OSI Reference Model - The ISO Model of Architecture for Open Systems Interconnection," **IEEE Trans. Commun.**, COM-28,4 (Apr. 1980), pp. 425-432.

[Zimme81] Zimmermann, H., "Progression of the OSI Reference Model and its Applications," **NTC**, pp. F8.1/1-6, 1981.

[Zloof81] Zloof, M.M., "QBE/OBE: A Language for Office and Business Automation," **IEEE Computer** 14,5 (May 1981), pp. 13-22.

[Zloof83] Zloof, M.M., "Clarifying Some Issues in Office Automation," **IEEE Database Engineering** 6,3 (Sept. 1983), pp. 4-11.

SUPPLEMENTAL BIBLIOGRAPHY

[Atwoo76] Atwood, J.W., "Concurrency in Operating Systems," **IEEE Computer** 9,10 (Oct. 1976), pp. 18-26. Also in [Liebo81, pp. 395-403].

[Bair78] Bair, J.H., "Communication in the Office-of-the-Future: Where the Real Payoff May be," **Proc. Fourth Int. Conf. on Comput. Commun.**, ICCC, pp. 733-739, 1978.

[Bair80] Bair, J.H., "Avoiding Working Non-Solutions to Office Communications System Design," **Compcon**, pp. 99-103, Spring 1980.

[Bair81] Bair, J.H., "Automated Office System Design: Problems and Principles," **NTC**, pp. G5.1/1-8, 1981.

[Bair83] Bair, J.H., "The Interface to Human-Computer Productivity in Office Systems," **Fourth Office Autom. Conf.**, pp. 21-23, 1983.

[Ball76] Ball, J.E., et. al., "RIG, Rochester's Intelligent Gateway: System Overview," **IEEE Trans. Softw. Eng.**, SE-2,4 (Dec. 1976), pp. 321-328.

[Barbe80] Barber, D.L.A., "A New Approach to the Evolution of Complex Protocols," in [Naffa80c, pp. 217-223].

[Bates75] Bates, D. (Ed.), Network Systems and Software, Infotech, England, (1975).

[Beauc81] Beauchamp, K.G. (Ed.), New Advances in Distributed Computer Systems, Reidel Publishing Co., Boston, (1981).

[Becke76] Becker, H.B., "The Distributed Environment - A Formal Structure for its Definition and Design," **Compcon**, pp. 364-366, Fall 1976.

[Belfo76] Belford, G.G., "Optimization Problems in Distributed Data Management," **Proc. Third Inter. Conf. Comput. Commun.**, (Aug. 1976), pp. 297-302.

[Bergl78] Berglund, R.G., "Comparing Network Architectures," **Datamation** 24,2 (Feb. 1978), pp. 79-85. Also in [Liebo81, pp. 92-97].

[Bernh82] Bernhard, R., "The Quandary of Office Automation," **IEEE Spectrum** 19,9 (Sept. 1982), pp. 34-39.

[Blanc74] Blanc, R.P., "Review of Computer Networking Technology," U.S. Government, Washington, (1974).

[Blanc76] Blanc, R.P., and Cotton, I.W. (Eds.), Computer Networking, IEEE Press, New York, (1976).

[Bochm80] Bochmann, G.V., and Sunshine, C.A., "Formal Methods in Communication Protocol Design," **IEEE Trans. Commun.**, COM-28,4 (Apr. 1980), pp. 624-631.

[Boggs80] Boggs, D.R., et. al., "Pup: An internetwork Architecture," **IEEE Trans. Commun.**, COM-28,4 (Apr. 1980), pp. 612-623.

[Bokha81] Bokhari, S.H., "Shortest Tree Algorithm for Optimal Assignments Across Space and Time in a Distributed Processor System," **IEEE Trans. Softw. Eng.**, SE-7,6 (Nov. 1981), pp. 583-589.

[Booth77] Booth, G.M., "Hierarchical Configurations for Distributed Processing," **Compcon**, Fall 1977.
Also in [Liebo81, pp. 246-247].

[Brign79] Brignoli, F., "Office Automation Planning within the Executive Office of the President," Conference Record, **Intern. Conf. on Commun.**, June 1979, pp. 22.1/1-3.
Also in [Thurb81, pp. 615-617].

[Brown82] Brown, J.G., "Selective Office Automation," **Computer Design** 14,12 (Dec. 1982), pp. 128-142.

[Burns77] Burns, J.C., "The Evolution of Office Information Systems," **Datamation** 23,4 (Apr. 1977), pp. 60-64.

[Carne79] Carne, E.B., "The Wired Household," **IEEE Spectrum** 16,10 (Oct. 1979), pp. 61-66.
Also in [Thurb81, pp. 605-610].

[Cashi76] Cashin, P.M., "Datapac Network Protocols," **Proc. Third ICCC**, (Aug. 1976), pp. 150-155.

[Caswe82] Caswell, S.A. (Ed.), Electronic Mail & Message Systems, 6,9 (May 3, 1982).

[Chand73] Chandra, A.N., "Some Considerations in the Design of Homogeneous Distributed Data Bases," **Compcon**, pp. 185-186, Spring 1973.

[Chatt83] Chattaway, A.T., and Chan, P.G., "Control System Integration: The Shape of Things to Come," **Intech** 30,9 (Sep. 1983), pp. 109-111.

[Chlam80] Chlamtac, I., et. al., "Performance Issues in Back-end Storage Networks," **IEEE Computer** 13,2 (Feb. 1980), pp. 18-31.

[Chu77] Chu, W.W. (Ed.), Advances in Computer Communications, Artech House, Massachusetts, (1977).

[Coffm81] Coffman, E.G. Jr., et. al., "Optimization of the Number of Copies in a Distributed Data Base," **IEEE Trans. Softw. Eng.**, SE-7,1 (Jan. 1981), pp. 78-84.

[Colli82] Collins, W., et. al., "A Network File Storage System," **Compcon**, pp. 20-23, Spring 1982.

[Craig80] Craig, L.C., "Office Automation at TI," **Compcon**, pp. 69-75, Fall 1980.

[Csaba81] Csaba, L., et. al., Networks From the User's Point of View, **Proc. IFIP TC-6 Working Conference COMNET'81**, May 1981.

[Datap82] Datapoint Office Automation Systems, **Comput. Equip. Rev.** 4,2 (July-Dec. 1982), pp. 129-137.

[Davis74] Davis, S., "A Fresh View of Mini- and Microcomputers," **Computer Design**, (May 1974), pp. 67-79. Also in [Liebo81, pp. 35-47].

[Day83] Day, J.D., and Zimmermann, H., "The OSI Reference Model," **Proc. of the IEEE** 71,12 (Dec. 1983), pp. 1334-1340.

[Deime80] Deimel, L.E., et. al., "A Case Study of a Distributed System Design," **Compcon**, pp. 636-642, Fall 1980.

[Denni78] Denning, P.J., "Operating Systems Principles for Data Flow Networks," **IEEE Computer** 11,7 (July 1978), pp. 86-96.

[Denni79] Dennis, J.B., "The Varieties of Data Flow Computers," **First Int. Conf. Data Flow Comput.**, IEEE, pp.430-439, 1979.

[desJa80] desJardins, R., and White, G.W., "ISO/ANSI Reference Model of Open Systems Interconnection," **Proc. Trends and Appl.**, IEEE (May 1980), pp. 47-53.

[desJa82] desJardins, R., "An OSI Architecture for Federal Government Information Systems," **Compcon**, pp. 76-91, Fall 1982.

[Eckho78] Eckhouse, R.H. Jr., and Stankovic, J.A., "Issues in Distributed Processing - An Overview of Two Workshops," **IEEE Computer** 11,1 (Jan. 1978), pp. 22-26. Also in [Thurb81, pp. 3-7].

[Edwar78] Edwards, G.C., "Organizational Impacts on Office Automation," **Proc. Fourth ICCC**, pp. 741-746, 1978.

[Elovi74] Elovitz, H.S., and Heitmeyer, C.L., "What is a Computer Network?," **IEEE 1974 NTC Record**, pp. NTC 74/1007-1014, 1974. Also in [Liebo81, pp. 169-176].

[Farbe72] Farber, D.J., "Networks: An Introduction," **Datamation** 18,4 (Apr. 1972), pp. 36-39.
Also in [Liebo81, pp. 188-191].

[Farbe73] Farber, D.J., et. al., "The Distributed Computing System," **Compcon**, pp. 31-34, Spring 1973.

[Farbe80] Farber, D.J., and Caine, S.H., "Computer Message Systems in the Office," **ICCC**, (Oct. 1980), pp. 43-50.

[Feng81] Feng, T., "A Survey of Interconnection Networks," **IEEE Computer** 14,12 (Dec. 1981), pp. 12-27.

[Fishe80] Fisher, M.L., and Hochbaum, D.S., "Database Location In Computer Networks," **Journ. of the ACM** 27,4 (Oct. 1980), pp. 718-735.

[Fletc73] Fletcher, J.G., "Octopus Communications Structure," **Compcon**, pp. 21-23, Spring 1973.

[Fortu77] Fortune, P.J., et. al., "Design and Implementation of a Local Computer Network," **ICC'77**, pp. 46.3/221-226.

[Fukud78] Fukuda, Z., "A Study of Centralized and Distributed Processing," **Proc. Fourth Int. Conf. on Comput. Commun.**, pp. 259-264, 1978.

[Gable80] Gabler, H.G., "Teletex - A New Approach Toward Office Automation," **ICCC**, pp. 121-126, (Oct. 1980).

[Gehri82] Gehringer, E.F., et. al., "The CM* Testbed," **IEEE Computer** 15,10 (Oct. 1982), pp. 40-53.

[Giffo80] Gifford, D.K., "Violet, an Experimental Decentralized System," in [Naffa80c, pp. 27-41].

[Gouda78] Gouda, M.G., "A Hierarchical Controller for Concurrent Accessing of Distributed Databases," **Fourth Workshop Comput. Arch. Non-Numeric Processing**, (Aug. 1978), pp. 65-70.

[Graha80] Graham, B.A., "High Level Protocols - Benifits and Implementations," **Data Commun. Intern.**, (June 1980), pp. 221-225.

[Graub82] Graube, M., "Local Area Nets: A Pair of Standards," **IEEE Spectrum** 19,6 (June 1982), pp. 60-64.

[Greni80] Greninger, L., and Roberts, R., "Considerations for Logical Virtual Terminal Interface," **Compcon**, pp. 33-40, Fall 1980.

[Hagen80] Hagen, R., "Internetting Techniques for Different Text Communication Services and the Interworking of Text Communication Networks," **ICCC**, (Oct. 1980), pp. 57-62.

[Hammo80] Hammond, R.A., "Experiences with the Series/1 Distributed System," **Compcon**, pp. 586-589, Fall 1980.

[Heins80] Heinselman, R.C., "System Design Selection for Distributed Data Systems," **Compcon**, pp. 203-210, Fall 1980.

[Hertw78] Hertweck, F., et. al., "X25 Based Process - Process Communication," **Computer Networks** 2,4/5 (Sept./Oct. 1978), pp. 250-270.

[Hevne79] Hevner, A.R., and Yao, S.B., "Query Processing in Distributed Database Systems," **IEEE Trans. Softw. Eng.**, SE-5,3 (May 1979), pp. 177-187.

[Hiltz78] Hiltz, S.R., and Turoff, M., "The Wired World of the Office of the Future: Applications and Impacts of Computerized Conferencing in the Multi-National Organization," **Proc. Fourth ICCC**, pp. 711-716, 1978.

[Hoare74] Hoare, C.A.R., "Monitors: An Operating System Structuring Concept," **Commun. of the ACM** 17,10 (Oct. 1974), pp. 549-557.

[Hobbs72] Hobbs, L.C., "Terminals," **Proc. of the IEEE**, pp. 1273-1284, 1972.
Also in [Liebo81, pp. 109-120].

[Holde80] Holden, J.B., "Solving Communication Problems - The Case for Electronic Mail," **Compcon**, pp. 65-68, Fall 1980.

[Horsl81] Horsley, J., "Power Where It's Needed," **Data Processing**, (Sept. 1982), pp. 33-35.

[Hosie80] Hosier, J., "Tackling the Problems of a Distributed Database," **Data Processing**, (Mar. 1980), pp. 22-23.

[Hqber80] Hqberecht, V.L., "Higher-Level Formats and Protocols in Systems Network Architecture," **ICC**, pp. 5.4/1-7, 1980.

[Huang82] Huang, H., "Performance Simulation of Local Area Distributed Systems," **First Annual Phoenix Conf. Comput. Commun.** (May 1982), pp. 369-373.

[Huen77] Huen, W., et. al., "A Network Computer for Distributed Processing," **Compcon**, Fall 1977.
Also in [Liebo81, pp. 248-252].

[Hwang82] Hwang, C.F., et. al., "A Database Server Architecture for Local Networks," **Compcon**, pp. 28-31, Spring 1982.

[Ivins81] Ivinson, J.L., and Jenkins, B.W., "Selecting a Distributed System," **Data Processing**, (Sept. 1981), pp. 24-26.

[Jones79] Jones, A.K., et. al., "StarOS, a Multiprocessor Operating System for the Support of Task Forces," **Proc. Seventh Symp. Oper. Syst. Prin.**, ACM, pp. 117-127, 1979.

[Kerr83] Kerr, I., and Steedman, D., "iNet Messaging and Protocols," Dept. 8E60, BNR-Meriline Court, 1983.

[Kesse81] Kessels, J.L.W., "The Soma: A Programming Construct for Distributed Processing," **IEEE Trans. Softw. Eng.**, SE-7,5 (Sept. 1981), pp. 502-509.

[Kirst78] Kirstein, P.T., and Yilmaz, S., "Facsimile Transmission in Message Processing System with Data Management," **Proc. Fourth ICCC**, pp. 717-726, 1978.

[Kumar79] Kumar, B., and Gonsalves, T.A., "Modelling and Analysis of Distributed Software Systems," **Proc. Seventh Symp. on Oper. Syst. Prin.**, ACM, pp. 2-8, 1979.

[Kutni79] Kutnick, D., "Distributed Processing and the Automated Office," **Proc. Intel Com 79**, 1979, pp. 56-59.

[Langs83] Langsford, A., et. al., "OSI Management and Job Transfer Services," **Proc. of the IEEE** 71,12 (Dec. 1983), pp. 1420-1424.

[Lann79] Lann, G.L., "An Analysis of Different Approaches to Distributed Computing," **First Inter. Conf. Distrib. Comput. Syst.**, Oct 1979, pp. 222-232.
Also in [Thurb81, pp. 40-50].

[LeLan80] Le Lann, G., "Control of Concurrent Accesses to Resources in Integrated Office Systems," in [Naffa80c, pp. 43-51].

[Lisko82] Liskov, B., "On Linguistic Support for Distributed Programs," **IEEE Trans. Softw. Eng.**, SE-8,3 (May 1982), pp. 203-210.

[Littl78] Little, J.L., "A Computer Network Protocol at the Application Level for Libraries and Other Information Science Services," **Journ. of Library Automation** 11,3 (Sept. 1978), pp. 239-245.

[Liu77] Liu, M.T., and Reames, C.C., "Message Communication Protocol and Operating System Design for the Distributed Loop Computer Network (DLCN)," **Fourth Annual Symp. on Comput. Arch.**, 77CH1182-5C (Mar. 1977), pp. 193-200.

[Li81] Li, V.O.K., "Performance Models of Distributed Databases," **NTC**, pp. G9.2/1-5, 1981.

[Lowen82] Lowenthal, E., "Database Systems for Local Nets," **Datamation** 28,8 (Aug. 1982), pp. 97-106.

[Macfa82] Macfarlane, D., "Framework Needed to Define What Integrated Systems Really Are," **Computer Data** 7,2 (Feb. 1982), pp. 36-39.

[Marti82] Martin, M.R., "UNIX and Local Computer Networking," Digest of Papers Spring Compcon 82, pp. 318-322.

[McCar82] McCartney, J., "Who will be in Control of Future Networks?," **Data Processing**, (Feb. 1982), pp. 18-19.

[Minou82] Minoura, T., and Wiederhold, G., "Resilient Extended True-Copy Token Scheme for a Distributed Data Base System," **IEEE Trans. Softw. Eng.**, SE-8,3 (May 1982), pp. 173-188.

[Moore83] Moore, D.J., "Teletex - A Worldwide Link Among Office Systems for Electronic Document Exchange," **IBM Syst. Journ.** 22/1-2 (1983), pp. 30-45.

[Myer80] Myer, T.H., "Future Message System Design: Lessons from the Hermes Experience," **Compcon**, pp. 76-84, Fall 1980.

[Murph83] Murphy, J.A., "Integrated Office-Automation Systems," **Mini-Micro Systems** 16,6 (May 1983), pp. 181-188.

[Naffa80a] Naffah, N., "An Approach to Message System Architecture," **Compcon**, pp. 165-169, Spring 1980.

[Naffa80b] Naffah, N., "An Experiment in the Design of a Public Message System," **Compcon**, pp. 137-140, Fall 1980.

[Naffa80c] Naffah, N. (Ed.), Integrated Office Systems, North-Holland, Amsterdam, (1980).

[Naffa82] Naffah, N. (Ed.), Office Information Systems, North-Holland, Amsterdam, (1982).

[Naffa83a] Naffah, N., "KAYAK: A Key to Office Automation," **Comput. Compacts** 1,1 (Feb. 1983), pp. 19-22.

[Naffa83b] Naffah, N., et. al., "The AGORA Message System Architecture," **Comput. Compacts** 1,1 (Feb. 1983), pp. 22-28.

[Newel80] Newell, A., et. al., "CMU Proposal for Personal Scientific Computing," **Compcon**, pp. 480-483, Spring 1980.

[Newma80] Newman, W., "Office Models and Office Systems Design," in [Naffa80c, pp. 3-10].

[Newma82] Newman, W.M., et. al., "Officetalk-Zero: An Experimental Integrated Office System," **Proc. of the European Conf.**, (1982), pp. 315-331.

[Niels82] Nielsen, F.H., and Moulton, J.R., "Description of a Planned Federal Information Processing Standard for File Transfer Protocol," **Infocom 82**, pp. 139-147, Spring 1982.

[Nutt81] Nutt, G.J., and Ricci, P.A., "Quinault: An Office Modeling System," **IEEE Computer** 14,5 (May 1981), pp. 41-58.

[Parr83] Parr, F.N., and Strom, R.E., "NIL: A High-Level Language for Distributed Systems Programming," **IBM Syst. Journ.** 22/1-2 (1983), pp. 111-126.

[Piatk79] Piatkowski, T.F., "A Formal Definition of the ISO-ANSI Open Systems Interconnection Reference Model -- A Brief Overview and Proposal for a State-Oriented Systems Approach," **First Int. Conf. Distrib. Comput. Syst.**, Oct. 1979, pp. 79-94. Also in [Thurb81, pp. 105-119].

[Picke73] Pickens, J.R., "Computer Networks from the User's Point of View," **Compcon**, pp. 71-74, Spring 1973.

[Picke79] Pickens, J.R., "Functional Distribution of Computer Based Messaging Systems," **Sixth Data Commun. Symp.**, Nov. 1979, pp. 8-17.
Also in [Thurb81, pp. 130-139].

[Pierc77] Pierce, R.A., and Moore, D.H., "Network Operating System Functions and Microprocessor Front-Ends," **Compcon**, pp. 325-328, Spring 1977.

[Pyke76] Pyke, T.N., "Assuring User Service Quality in a Distributed Computer Network," **Compcon**, Spring 1976.
Also in [Liebo81, pp. 197-200].

[Raag81] Raag, H., "International Electronic Mail," **NTC**, pp. A9.1/1-5, Fall 1981.

[Redel80] Redell, D.D., et. al., "Pilot: An Operating System for a Personal Computer," **Commun. of the ACM** 23,2 (Feb. 1980), pp. 81-92.

[Ries82] Ries, D.R., and Smith, G.C., "Nested Transactions in Distributed Systems," **IEEE Trans. Softw. Eng.**, SE-8,3 (May 1982), pp. 167-172.

[Rosen80] Rosen, B., "PERQ: A Commercially Available Personal Scientific Computer," **Compcon**, pp. 484-485, Spring 1980.

[Russe82] Russell, W.H., et. al., "The Electronic Document Network - A Prototype Distributed Information System for the Navy," **Compcon**, pp. 362-365, Spring 1982.

[Ryle82] Ryle, B.L., "Delta Specific/General Purpose System," **Proc. Interface '82**, pp. 4-9, 1982.

[Saal83] Saal, H., "Local Area Networks - An Update on Microcomputers in the Office," **BYTE** 8,5 (May 1983), pp. 60-78.

[Sabla81] Sablatash, M., and FitzGerald, R., "Towards the Design of an ISO OSI Layered Architecture for a Canadian Broadcast Telidon System," **NTC**, pp. B2.4/1-8, Fall 1981.

[Saito78] Saito, M., et. al., "Concept and Design Considerations of Hierarchical Database Access Protocol for Distributed Database," **Proc. Fourth ICCC** (Sept. 1978), pp. 817-822.

[Schic76] Schicker, P., and Duenki, A., "Network Job Control and its Supporting Services," **Proc. Third Inter. Conf. Comput. Commun.**, (Aug. 1976), pp. 303-307.

[Schin81] Schindler, S., et. al., "Open Systems Interconnection - the Presentation Service Model," **Comput. Commun. (GB)** 4,5 (Oct. 1981), pp. 227-241.

[Schin82] Schindler, S., "The New ISO Standards for Communications and Office Automation," **Computer Networks** 6,4 (Sept. 1982), pp. 291-298.

[Schne80] Schneider, M.L., "The Application of Human Factors to Distributed Systems," **Compcon**, pp. 117-118, Fall 1980.

[Schoe78] Schoemaker, S. (Ed.), Computer Networks and Simulation, North-Holland, New York, (1978).

[Schul83] Schultz, G., "An Intelligent-Workstation Approach to Office Automation," **Mini-Micro Systems** 16,6 (May 1983), pp. 193-200.

[Schul78] Schulze, G., and Borger, J., "A Virtual Terminal Protocol Based Upon the 'Communication Variable' Concept," **Computer Networks** 2,4/5 (Sept./Oct. 1978), pp. 291-296.

[Schwa77] Schwartz, M., Computer Communication Network Design and Analysis, Prentice-Hall, New Jersey, (1977).

[Shaw74] Shaw, A.C., The Logical Design of Operating Systems, Prentice-Hall, New Jersey, (1974).

[Shoch82a] Shoch, J.F., and Hupp, J.A., "The 'Worm' Programs - Early Experience with a Distributed Computation," **Commun. of the ACM** 25,3 (Mar. 1982), pp. 172-180.

[Shoch82b] Shoch, J.F., et. al., "Evolution of the Ethernet Local Computer Network," **IEEE Computer** 15,8 (Aug. 1982), pp. 10-27.

[Shu82] Shu, N.C., et. al., "Specification of Forms Processing and Business Procedures for Office Automation," **IEEE Trans. Softw. Eng.**, SE-8,5 (Sept. 1982), pp. 499-512.

[Simps78] Simpson, R.O., and Phillips, G.H., "Network Job Entry Facility for JES2," **Proc. Fourth ICCC**, pp. 265-270, 1978.

[Sinco80] Sincoskie, W.D., and Farber, D.J., "The Series/1 Distributed Operating System: Description and Comments," **Compcon**, pp.579-584, Fall 1980.

[Sloni81] Slonim, J., et. al., "NDX-100: An Electronic Filing Machine for the Office of the Future," **IEEE Computer** 14,5 (May 1981), pp. 24-36.

[Smith79] Smith, R., "The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver," **Proc. First Int. Conf. Distrib. Comput. Syst.**, IEEE, pp. 185-192, 1979.

[Solnt81] Solntseff, N., "Artemis - An Operating System for OSI Microcomputers," **Sigsamll Symp. on Small Syst.** and **Sigmod Workshop on Small Database Syst.**, (Oct. 1981), pp. 30-33.

[Spect81] Spector, A.Z., "Performing Remote Operations Efficiently on a Local Computer Network," **Eighth Symp. Oper. Syst. Principles**, (Dec. 1981), in **Commun.** ACM 25,4 (Apr. 1982), pp. 246-260.

[Stack80] Stack, T.R., and Dillencourt, K.A., "Protocols for Local Area Networks," **Proc. Trends and Appl.**, IEEE (May 1980), pp. 83-93.

[Stuck78] Stucki, M.J., et. al., "Coordinating Concurrent Access in a Distributed Database Architecture," **Fourth Workshop on Comput. Arch. for Non-Numeric Processing** (Aug. 1978), pp. 60-64.

[Su80] Su, S.Y.W., et. al., "Database Machines and Some Issues on data base system Standards," **NCC**, pp. 191-208, 1980.

[Swine79] Swinehart, D., et. al., "WFS: A Simple Shared File System for a Distributed Environment," **Proc. Seventh Symp. on Oper. Syst. Prin.**, ACM, pp. 9-17, 1979.

[Sykes80] Sykes, D.J., "The Economics of Distributed Systems," **Compcon**, pp. 8-15, Fall 1980.
Also in [Liebo81, pp. 20-27].

[Tapsc80] Tapscott, D., et. al., "Towards a Methodology for User-Driven Design of Electronic Office Systems," **Compcon**, pp. 129-135, Fall 1980.

[Tapsc81] Tapscott, D., "Investigating the Office of the Future," **Telesis** 8,1 (1981), pp. 2-6.

[Tapsc82] Tapscott, D., "How Does the Office of the Future Measure Up?," **Teleph. Eng. and Manage.** 86,1 (Jan. 1982), pp. 50-52.

[Terad80] Terada, M., et. al., "A Network Operating System for High Speed Optical Fiber Loop Transmission System," **ICCC**, (Oct. 1980), pp. 641-646.

[Thoma80] Thomas, R.T., "Distributed Computing in a University Environment," **Compcon**, pp. 643-644, Fall 1980.

[Toong77] Toong, H.D., "Microstar, A Microprocessor Controlled Distributed Minicomputer Network," **Compcon**, pp. 320-324, Spring 1977.

[Unsoy81] Unsoy, M.S., "Performance Monitoring and Evaluation of Datapac Network," **NTC**, pp. F6.5/1-5, 1981.

[Uhlig79] Uhlig, R.P., et. al., The Office of the Future, North-Holland, Amsterdam, (1979).

[Wang80] Wang, P.S.C., and Kimbleton, S.R., "A Data Transfer Protocol for Remote Database Access," **Proc. Trends and Appl.**, IEEE (May 1980), pp. 70-82.

[Ward80a] Ward, S.A., "An Approach to Personal Computing," **Compcon**, pp. 460-465, Spring 1980.

[Ward80b] Ward, S.A., "TRIX: A Network-Oriented Operating System," **Compcon**, pp. 344-349, Spring 1980.

[Watso82] Watson, I., and Gurd, J., "A Practical Data Flow Computer," **IEEE Computer** 15,2 (Feb. 1982), pp. 51-57.

[White77] White, R.B., "A Prototype for the Automated Office," **Datamation** 23,4 (Apr. 1977), pp. 83-90.

[Wilcz80] Wilczynski, D., et. al., "Message System Architecture - Experience at CINCPAC with the SIGMA System," **Compcon**, pp. 170-173, Spring 1980.

[Wimme80] Wimmer, K.E., "An Architecture for Office Systems," **ICCC**, (Oct. 1980), pp. 133-138.

[Witti79] Wittie, L.D., "A Distributed Operating System for a Reconfigurable Network Computer," **Proc. First Int. Conf. on Distrib. Comput. Syst.**, IEEE, pp. 669-677, 1979.

[Yau81] Yau, S.S., et. al., "An Approach to Distributed Computing System Software Design," **IEEE Trans. Softw. Eng.**, SE-7,4 (July 1981), pp. 427-435.

INDEX OF REFERENCES

INDEX OF DEFINITIONS