SNA Distribution Services

by B. C. Housel
C. J. Scopinich

This paper describes the IBM SNA Distribution Services (SNADS). Heretofore, SNA has focused on synchronous data distribution. Along with the advent of office systems and other distributed applications has come the requirement to provide a common architecture for interchanging data asynchronously among diverse systems and products. SNA Distribution Services provides a general asynchronous (delayed delivery) data distribution facility for SNA applications. The initial implementations are for office systems applications. Discussed are objectives for an asynchronous data distribution service, key architectural concepts, the relationship between SNADS and the SNA synchronous communication architecture, and the interface between the distribution service and application transaction programs.

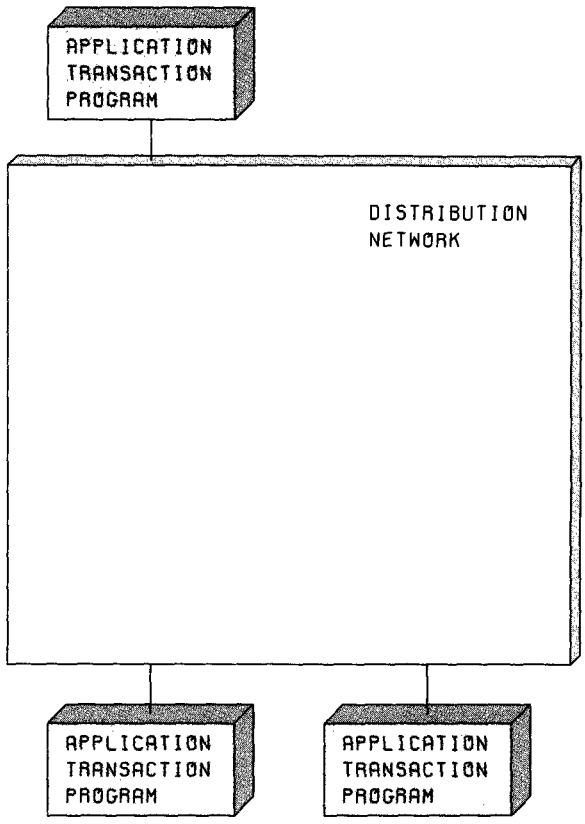
Although SNA has previously been directed primarily towards synchronous data communication, with the advent of office systems and other distributed applications has come the need for common architecture for interchanging data asynchronously among diverse systems and products. This paper describes the IBM SNA Distribution Services (SNADS),[1] which is intended to satisfy this need.

SNA Distribution Services is provided by a set of architected transaction programs that run in a Logical Unit Type 6.2 (LU 6.2) environment using the SNA Advanced Program-to-Program Communication (APPC) services.[2,3] LU 6.2 is used for the synchronous transmission of distributions in an SNADS network. Further details on the use of LU 6.2 in SNADS are given in the section on synchronous communication between distribution service units later in this paper.

SNADS is designed to be a general service, usable by any application. Its development has been motivated by the fact that asynchronous data distribution is required by many distributed applications and systems services, including office systems, network management, file transfer, and job networking. The lack of a common, general architecture for asynchronous data distribution results in application-, system-, or product-specific variations of the same function. This inhibits data interchange, results in reduced utilization of network resources, and increases costs due to duplicated design and implementation. The initial implementations of SNA Distribution Services are for office systems applications. The implementing products are DISOSS Version 3.2 and the IBM 5520 Release 5.

SNA Distribution Services is provided by interconnected *distribution transaction programs*[4] that cooperate to perform *asynchronous data distribution.* The set of distribution transaction programs and their interconnections form an SNA Distribution Services network (or more simply a "distribution network"). As shown in Figure 1, application transaction programs interface to the distribution network to make requests to send or receive distributions[5] from the distribution service. An origin application transaction program requests the dis-

Figure 1 Application transaction programs connected to a distribution network



tribution service to send a distribution to one or more destination application transaction programs.

Asynchronous versus synchronous communication. To understand the nature of SNA Distribution Services, it may be useful to contrast synchronous and asynchronous communication. Two common systems that illustrate the difference are the telephone system and the postal system. The telephone system provides synchronous communication, and the postal system provides asynchronous communication.

Synchronous communication requires the sender and receiver to converse in real time. This means that the sender, the receiver, and the communications resources must be active simultaneously. Typically, in synchronous communication the parties alternately "talk" to send data and "listen" to receive data. A message sent by one party solicits an immediate response from the other party.

With asynchronous communication, the sender may submit a request (such as a person mailing a letter) without participation or even knowledge on the part of the receiver. In SNADS, an origin application transaction program may request a distribution without an active destination application transaction program. After the distribution service has accepted responsibility for the distribution request, the origin application transaction program may terminate. With asynchronous communication, requests are queued (like a letter stored in a mail box) and processing proceeds as resources become available. The resources required for a distribution may be available only at certain times. Thus, depending on the route that a distribution travels and other factors regarding such handling instructions as priority specified in the request, data may be delayed at intermediate points along the route. This means that the time it takes for a distribution to travel from its origin to a destination may vary greatly for different requests (i.e., seconds, minutes, days, etc.). For this reason, an asynchronous data distribution service is sometimes referred to as a "delayed delivery service."

With asynchronous communication, when a distribution arrives at its destination, the communication service maintains responsibility for the data until the receiver can be located—as in registered mail, by analogy—or until it is convenient for the receiver to take delivery of the distribution, such as the person who may open his post office box any time he wishes. In SNA Distribution Services, a distribution is queued at the destination and is delivered to the destination application program either upon arrival or at the convenience of the destination application program.

As in synchronous communication, asynchronous communication may require responses to requests (e.g., confirmation of delivery). Synchronous communication has the property that responses are synchronized with requests. That is, a response to a message must be received before another message can be sent. Asynchronous communication has the property that responses resulting from one or more distribution requests from a common origin may be returned in any order. Thus, some means of correlating responses with the requests must be defined. Because correlation requirements vary considerably by application, this function is an application responsibility and not part of SNADS. The SNADS architecture, however, defines a unique distribution identifier that makes correlation possible and pro-

vides a means for reporting notification (status) information. Further details on how responses may be correlated with requests are given later, in the section on notification facilities.

General design objectives and requirements

The key design objectives in developing the SNA Distribution Services architecture have been application independence, ease of use, manageability, efficiency, and extendability.

Application independence. An application-independent architecture must satisfy a broad spectrum of application requirements. A general distribution service must be insensitive to the type of data being transported. It should be possible to distribute any bit stream—documents, files, digitized audio, etc. Provisions must exist to handle a wide range of distribution sizes efficiently, from small messages to voluminous bulk data. Users must be able to name explicitly the programs that are to store the data and receive the distributions at the destinations. Different applications require the distribution service to honor different handling instructions for each type of given request. Some applications require the ability to distribute information to multiple destinations (recipients) in a single distribution. Applications differ in their requirements for notification regarding the status of distribution requests.

Ease of use and manageability. It should be possible to make changes in the distribution network with minimal disruption to users, applications, and system administrators. For example, the addition, deletion, or movement of users or network configuration changes should be localized to the affected entities in the network. SNA Distribution Services applications should be insulated from details of the distribution service to avoid affecting application code when changes occur in the distribution service.

Efficiency. A key goal in communications architectures is efficiency, because communications resources are valuable. There are, however, additional considerations in an asynchronous data distribution service. It is necessary to make efficient use of storage (e.g., disk space) and to minimize storage access. For example, in SNA Distribution Services, data are accessed directly at the origin and destinations of a distribution, in contrast to requiring a spooling technique. Another objective is to route distributions efficiently within the distribution network to reduce the amount of data transported and to provide a satisfactory level of service. Efficient routing is complicated in SNA Distribution Services because a single distribution request may contain multiple destinations.

As new requirements arise, SNA Distribution Services must be extended.

Extendability. As new requirements arise, SNA Distribution Services must be extended. Mechanisms to cope with extensions must be built in at the outset if graceful evolution of the architecture is to occur. One example of this is the use of self-defining constructs for enveloping the information distributed in the network.

The remainder of this paper focuses on the major concepts of the SNA Distribution Services architecture. The overall structure of the architecture and the basic terminology are introduced, each major concept is presented, and various design decisions are discussed.

Major concepts and terminology

A *Distribution Service Unit* (DSU) provides the distribution service to application transaction programs. A DSU comprises the distribution transaction programs that execute in a logical unit (LU) of an SNA node. As shown in Figure 2, a *distribution* network consists of a collection of interconnected DSUs. A line connecting two DSUs depicts a *distribution* connection. A distribution connection means that there are potential synchronous SNA sessions that can be used for synchronous communication between DSUs. A distribution connection exists regardless of whether there are any *active sessions* at any particular time. Also shown in Figure 2 are the different roles that application transaction programs and DSUs may assume—origin, intermediate, and destination.

Figure 2 Interconnected DSUs and application transaction programs

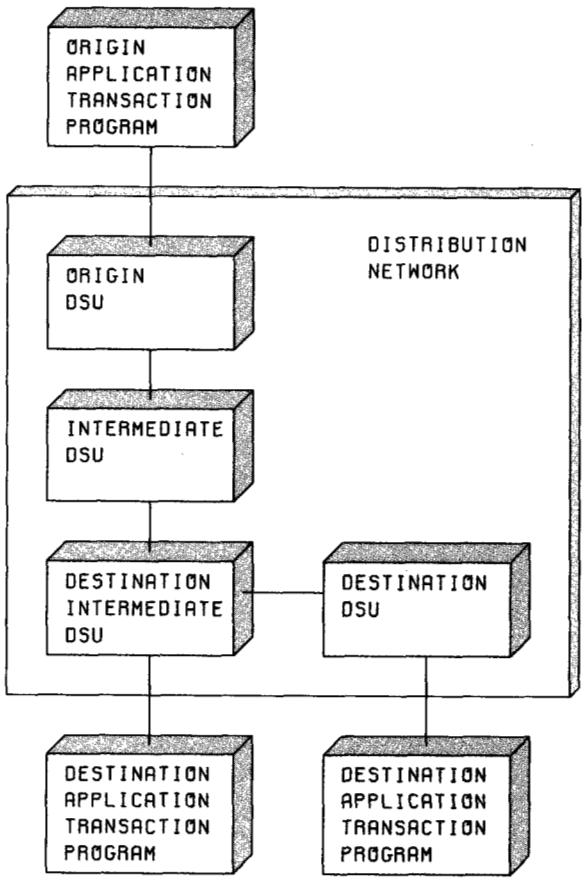
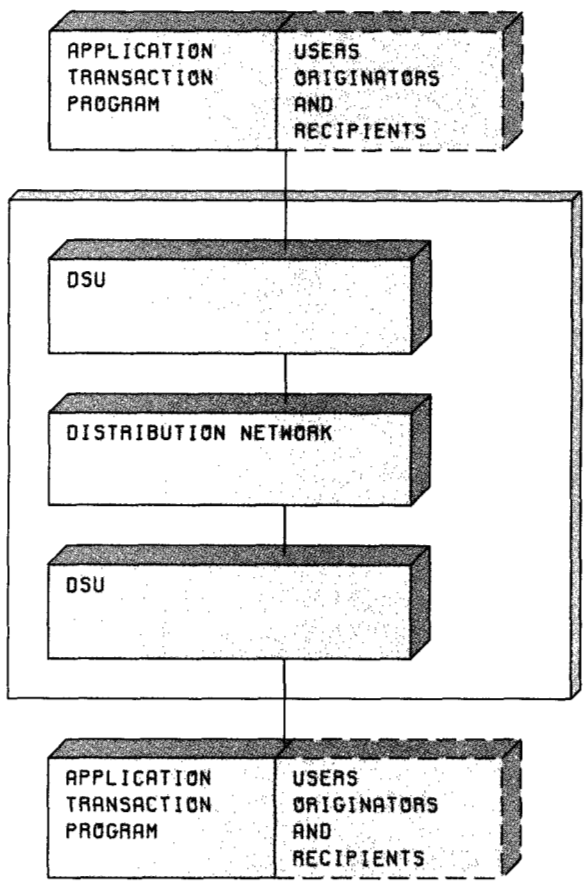


Figure 3 shows that *application transaction programs* serve as agents that operate on behalf of users of the *distribution service*. User facilities are located at a DSU, which corresponds to a set of resources in the SNA node (e.g., queues, files, access privileges, etc.). Each user is uniquely identified in a distribution network.[6] The *origin application transaction program* calls the distribution service to initiate a distribution on behalf of an originator, and the *destination application transaction program* calls the distribution service to receive distributions that have been queued for recipients.

An SNADS user is usually a person who is using a distribution network, as in the case of an office system. A user, however, may be such other types of entities as a device or a subsystem. In the examples in this article, users are persons, although the network is unaware of whether the user name refers to a person, device, or subsystem.

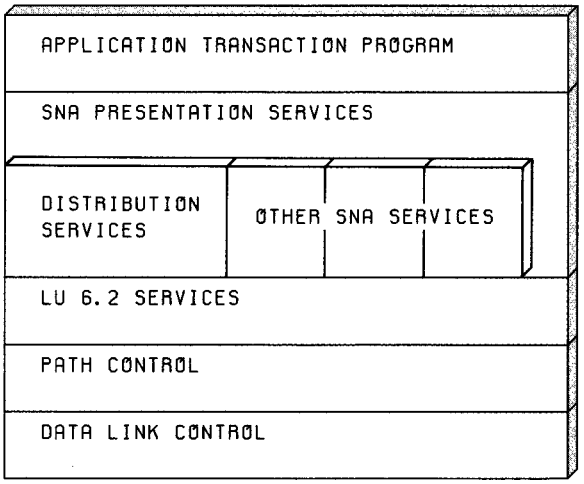
SNA Distribution Services uses the synchronous SNA services provided by LU 6.2. A DSU consists of architected transaction programs that run in an LU 6.2 environment. Figure 4 shows the relationships among an application transaction program and the distribution service and other SNA layers. SNA defines a logical interface, termed a *protocol boundary*,[7] between application transaction programs and the services provided by SNA. The protocol boundary definition is described as a set of verbs. The LU 6.2 protocol boundary consists of those verbs defined to use the LU 6.2 services.[3] Similarly, the distribution protocol boundary consists of the following verbs defined for using distribution services: DISTRIBUTE_DATA, DISTRIBUTE_STATUS,

Figure 3 Application transaction programs and distribution users



and RECEIVE_DISTRIBUTION. The later section on notification facilities illustrates the use of these verbs by application transaction programs. As shown in Figure 4, an application transaction program may use the distribution services and/or the LU 6.2 services, depending on its needs. The distribution transaction programs use the LU 6.2 verbs, as required for synchronous communication between adjacent DSUs, for example, to send a distribution. Another important protocol boundary required by the distribution service is the server protocol boundary, which is used to access storage in order to retrieve and store the object of a distribution. Servers are discussed in greater detail in a separate section later in this paper.

Figure 4 A layered view of distribution services in SNA



Message units. An important aspect of a communications architecture is the definition of the *message units* or *envelopes* that are used to carry information from one process to another. In SNADS, these envelopes are called *interchange units*. Interchange units use the general encoding structure defined by the Document Interchange Architecture (DIA)[8,9] for its document interchange units. Interchange units are architected as self-defining data streams. This allows the architecture to be easily extended with minimal effect on coding. The design of self-defining data streams is discussed at length in Reference 10.

Currently, two interchange units are defined for distribution. A *distribution interchange unit* is used to send a distribution to an adjacent DSU, and an *acknowledge interchange unit*[11] is used to report exceptions. When a DSU encounters an error while receiving a distribution interchange unit, it sends a *negative acknowledge interchange unit* to the sending DSU. The major components of the SNADS interchange units are shown in Figure 5. The PREFIX and SUFFIX delimit the interchange unit, and the COMMAND contains the control information necessary to perform the requested function. The DISTRIBUTE COMMAND contains such information as names and addresses of those who are to receive the distribution and the name of the destination

Figure 5 Distribution interchange units

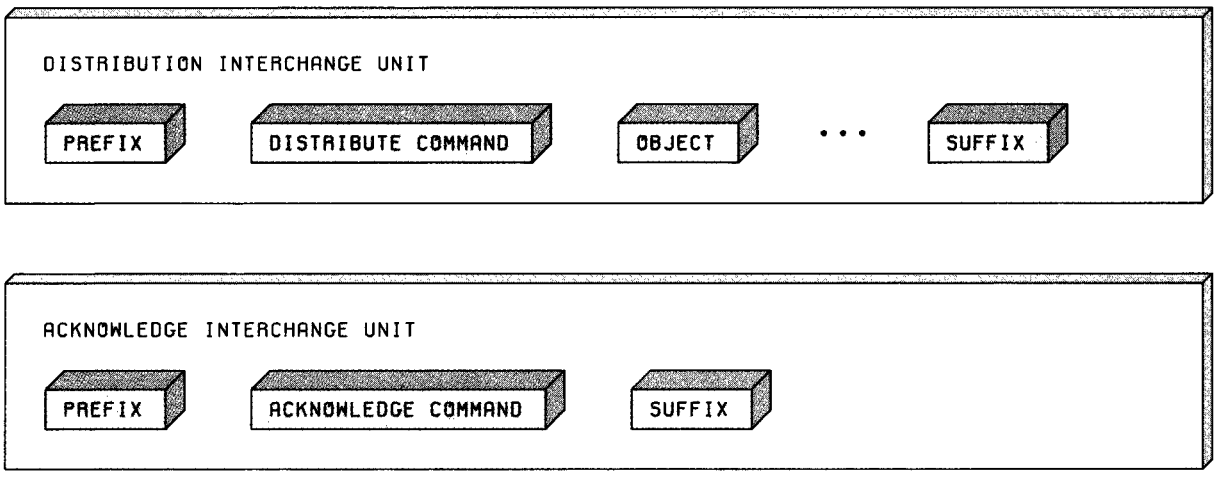
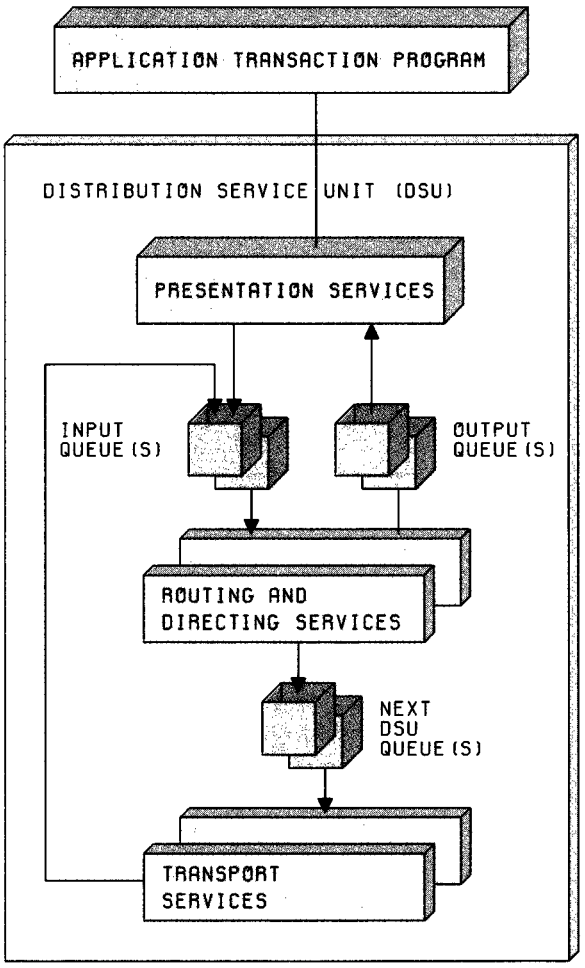


Figure 6 Structure of a Distribution Service Unit



transaction program. The OBJECT in the distribution interchange unit envelopes the contents to be distributed. A distribution interchange unit may contain zero or more objects. A user may send zero objects if it is desired simply to invoke a destination transaction program with a parameter string that is carried in the COMMAND. As an option, more than one object may be distributed.

The SNADS model. In order to describe the functions of the SNADS architecture, we define an SNADS reference model of a *Distribution Service Unit* (DSU);[12] Figure 6 shows its major components.

Presentation services process the distribution protocol boundary verbs issued by the application transaction program.[13] Part of this processing includes parameter checking and mapping the verb parameters to an internal form. A distribution request causes an entry to be added to the input queue. A request to receive a distribution causes an entry to be dequeued from an output queue if the queue is not empty. The dequeued entry is mapped into parameters for the application transaction program, and control is subsequently returned to the transaction program.

Routing and directing services process each entry on the input queue. This includes determining the addresses of the recipients of the distribution, if necessary, routing the distribution to the appropriate distribution queue(s) for the next phase of processing, and starting the appropriate transaction programs to process the distribution queues. Distributions are enqueued on output queues for local recipients and on next-DSU queues for remote recipients. The analysis by routing and directing services of an entry retrieved from the input queue may spawn several entries, each of which is enqueued on a different distribution queue.

Transport services use the facilities of LU 6.2 to transfer distributions between adjacent DSUs. To send a distribution to another DSU, transport services process entries from the next DSU queue(s). In receiving distributions, transport services build a queue entry and enqueue it on the input queue to await further processing by routing and directing services. Transport services use the server protocol boundary verbs as required to access (read/write) distribution objects in storage.

Architectural concepts

This section describes the primary concepts in the SNADS architecture: naming and addressing, the distribution service level, synchronous communication between Distribution Service Units, the SNADS notification facilities, and servers.

User naming and addressing. Naming and addressing are the foundation of any communications architecture. A *user name* identifies the user, and a *user address* identifies user's location. The naming and addressing design for SNADS is motivated by the following objectives:

- Users and application transaction programs should be insulated as much as possible from changes in the distribution network. It should be

possible for a user to move from one DSU to another without having to notify all other users of the change.

- The architecture should allow name management to be either centralized or distributed.
- Because SNADS must satisfy the requirements for a wide range of products, the architecture should not require all distribution service units to support large tables.

These considerations have led to the design of the user naming and addressing concepts. The naming of users should not depend on the topology of the distribution network; thus, separate *name spaces* are defined for users and Distribution Service Units. A user is named using a Distribution User Name (DUN), and a Distribution Service Unit is named using a Distribution Service Unit Name (DSUN). The name of the DSU where a given user is located is the user's address. At any instant in time, a user has only one address. A Distribution Service Unit Name is generally defined as a two-part, hierarchical name consisting of a Routing Group Name (RGN) and a subordinate Routing Element Name (REN). The routing group name is used to group Distribution Service Units topologically. This facilitates the use of smaller routing tables and decentralized naming of Distribution Service Units. Figure 7 shows three routing groups named RALEIGH, AUSTIN, and SANJOSE. Currently, the routing group name is optional, so that a DSU name may be defined as either a one-part or a two-part name.[14] The remainder of this paper assumes a single-part Distribution Service Unit name. A distribution user name is defined as a two-part, hierarchical name consisting of a Distribution Group Name (DGN) and a subordinate Distribution Element Name (DEN). Thus, DGN.DEN comprises a fully qualified, network-unique name of a user. The Distribution Group Name can be used to define such logical user groups as departments or divisions. For a given distribution group name, all distribution element names must be unique.

Figure 8 shows the relationships between user names and DSU names. The dashed-line boxes depict various distribution groups and distribution elements within individual distribution groups. For illustrative purposes, assume that a company has defined distribution groups for the functional areas of manufacturing (MAN), payroll (PAY), engineering (ENG), and personnel (PER). The Distribution Element Names (DENs) specify the names of employees in the respective groups. Examples of

Figure 7 Routing groups

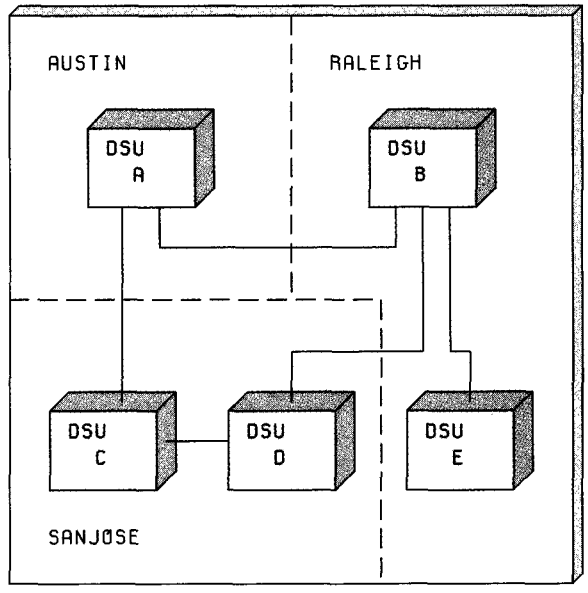


Figure 8 Relationships of users and DSUs

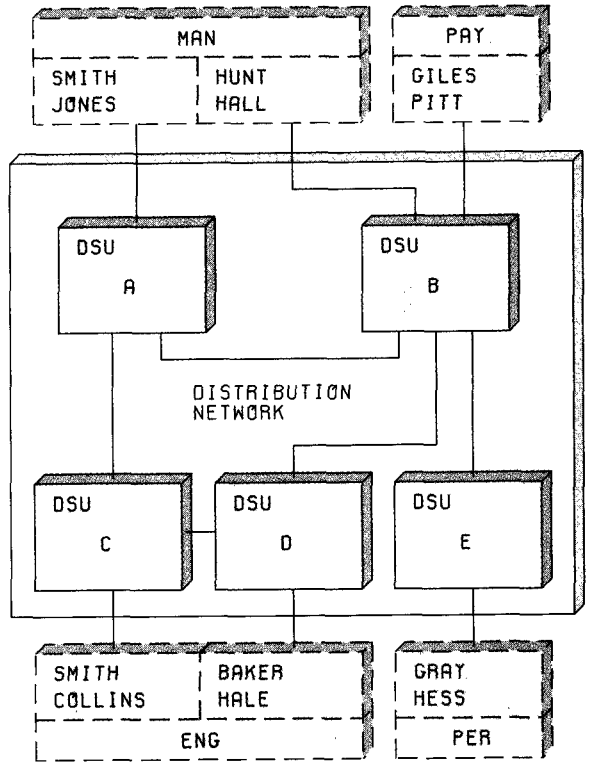


Table 1 Complete distribution dictionary

USER NAME (KEY)	DSUN
PER.GRAY	E
PER.HESS	E
ENG.BAKER	D
ENG.COLLINS	C
ENG.HALE	D
ENG.SMITH	C
MAN.HALL	B
MAN.HUNT	B
MAN.JONES	A
MAN.SMITH	A
PAY.GILES	B
PAY.PITT	B
.	ERROR

Table 2 Directories with default entries

AT DSU B:

USER NAME (KEY)	DSUN
PER.*	E
ENG.*	D
MAN.HALL	B
MAN.HUNT	B
MAN.JONES	A
MAN.SMITH	A
PAY.GILES	B
PAY.PITT	B
.	ERROR

AT DSU C:

USER NAME (KEY)	DSUN
PER.GRAY	E
PER.HESS	E
PER.*	ERROR
.	B

user names are ENG.HALE and MAN.HUNT.[15] The distribution network in Figure 8 shows five DSUs with DSU names A, B, C, D, and E.

The relationship of a distribution group to a DSU may be on a one-to-one basis (e.g., PER to E and PAY to B), or on a one-to-many basis (e.g., ENG to {C, D} and MAN to {A, B}). Any number of distribution groups may be defined at a single DSU. (Notice that both PAY and MAN are defined at DSU B.) As long as a user's name remains the same, a user may be moved to another DSU (i.e., incur an address change) with no impact to other distribution users. The distribution service then routes distributions to the new address (i.e., the new DSU) transparently to the users. Further details are described later in this paper in the section on routing and directing services.

The independence of naming of users and DSUs requires that routing and directing services resolve user names into destination DSU names. Each DSU maintains a distribution directory for this purpose. Table 1 shows a complete directory for Figure 8 that contains an entry for each user. For each distribution request, routing and directing services obtain the DSUN for each respective user name in the request. The destination DSUNs are used in routing the distribution.

An important design consideration has been to devise a means whereby some DSUs may maintain only a subset of the complete directory. In general, forcing all DSUs to maintain a complete directory could result in poor utilization of storage and increased directory maintenance cost. Moreover, this overhead might preclude smaller products from implementing the SNADS architecture. To remedy this problem, SNADS allows directories to contain default entries.

Table 2 illustrates how the directories with default entries might look for DSU E and DSU B in Figure 8. The character "*" denotes a default entry. This means that a comparison of any value results in the value TRUE. The directory for DSU E is simple. Except for the local recipients, all distributions designate DSU B as the destination DSU. Note that the directory contains entries for the local users as well as for remote users. When a default match is found for a recipient name, the distribution is routed to the associated address (destination DSUN). Upon arriving at the default DSU, another address may be determined for the recipient, in which case the distribution will be routed to the new address. This can occur several times until the distribution arrives at the recipient's true address or until a routing error is detected. For example, using the directories in Table 2, suppose a distribution request was made at DSU E to distribute data to the ENG.SMITH. At DSU E, ENG.SMITH maps to the DSUN of B. When the distribution arrives at DSU B, ENG.SMITH is mapped to a new destination DSUN, D. Finally, at DSU D, ENG.SMITH is resolved to C, the true address. Using defaults, a distribution may take a more indirect route than if the real address

were known at the outset. The extent to which defaults are to be used is determined by the distribution network administrator. The design of distribution directories is beyond the scope of this paper. We simply caution that defaults should be chosen carefully to avoid looping of distributions in the network.

The ability of DSUs to redirect traffic, as just described, enables users to move from one DSU to another with no disruption of service. Initially, the directory at the old address is updated to reflect the new address, and the user is defined as a recipient at the new location. The other directories in the distribution network may be gradually updated over time without losing data, because all distributions sent to the user's old address are redirected to the new address.

For each capability, a particular level of service may be specified.

Distribution service level (DSL). The purpose of DSL is to permit users to specify how a distribution is to be handled by the distribution service. Its role in a distribution network is similar to that of class of service in a synchronous SNA network.[16] DSL is supplied by the origin application transaction program in the distribution request, and specifies a list of capabilities that the user requires of the distribution service to meet the requirements of the distribution request. For each capability, a particular level of service may be specified. Currently, the following three capabilities have been architected.[17]

PRIORITY specifies the urgency with which the distribution service is to handle a distribution relative to other distributions in a distribution network. Currently, for general data distribution, two priority values (HIGH and LOW) have been defined. There are also the following two special priorities for expedited distributions: FAST for data distributions, and STATUS for status distributions. These priorities are higher than either of the priorities for general data distribution. Typically, distributions with a priority level of FAST contain small objects and, therefore, contain a small capacity level. Status distributions are used to return status information about distribution requests and are discussed further in the section on notification facilities.

CAPACITY specifies the level of storage capacity that all DSUs must provide along a route that a distribution travels from the origin DSU to the destination DSUs. This capability is provided to guard against sending distributions that contain very large objects to DSUs that have inadequate storage capacity. One value for this parameter is INDEFINITE, which means that the object size is not known at the origin DSU. DSUs that support this value always attempt to store the object.

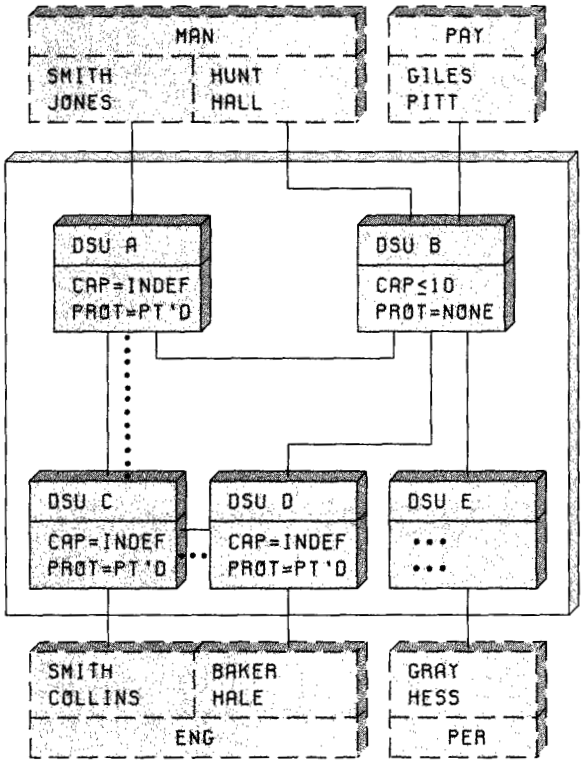
PROTECTION specifies the level of storage protection that all DSUs must provide along a route that a distribution is to travel from the origin DSU to the destination DSUs. Currently, the two levels of protection architected are PROTECTED and NONE. PROTECTED means that a DSU protects a distribution against communication and system failures (except for media failures such as a disk head crash). NONE means that distributions may be lost in the event of communication or system failures. This option allows some distributions to bypass the overhead of storing objects in a protected store, thus improving the throughput and allowing a greater number of feasible routes for a distribution, at the expense of distribution integrity.[18]

DSL is used extensively in the following ways by the distribution service in processing distributions:

- Selecting the next DSU of a distribution en route to a destination DSU.
- Selecting the type of session to be assigned to the LU 6.2 conversation used in sending a distribution to the next DSU.
- Selecting the appropriate next-DSU queue in order to properly schedule transmission according to the requested priority.
- Selecting the appropriate output queues at the destination DSUs.

Given a DSL, the task of the distribution service is to select a route that provides sufficient storage and communications resources to satisfy the request. Figure 9 shows the storage capabilities of various DSUs and communication capabilities to handle high-priority and low-priority traffic. The lines

Figure 9 DSUs with different capabilities



between a pair of DSUs denote distribution connections that are defined to handle high priority (solid line) and low priority (dotted line) traffic.

Suppose a distribution is requested from ENG.HALE (at DSU D) to MAN.JONES (at DSU A), with a DSL that specifies a priority level of LOW, a protection level of PROTECTED, and a capacity level of INDEFINITE. Although there are two possible routes between DSU D and DSU A (one through DSU C and one through DSU B), the only feasible route for this request is through DSU C. First, DSU B does not provide adequate object protection or capacity. In addition, the synchronous communications through DSU B are not configured to handle low-priority traffic. Thus DSL pertains to the complete route that a distribution travels. The use of DSU is constrained initially to a limited number of combinations of the above capabilities.

The DSL facility is designed to accommodate large networks that require sophisticated routing. Some of the design considerations are presented here. Each capability in the DSL is carried in a distribution. This is in contrast to the approach of using a single field to specify multiple capabilities, as was done with the SNA class of service. The latter approach is preferable when the total (combined) number of capabilities is not excessive. In a large distribution network, it is expected that the spectrum of capabilities may be extensive. For example, suppose there are ten priority levels, ten capacity levels, and three protection levels defined for a distribution network. This leads to the possibility of 300 (i.e., $10 \times 10 \times 3$) routing table entries for a single destination DSU. The decision to maintain discrete capabilities enables DSUs to make routing decisions algorithmically in order to keep routing tables as small as possible. Using this example, the number of routing table entries reduces to 23 (i.e., $10 + 10 + 3$) entries when a DSU implementation is designed to process each capability separately.

Routing and directing services. The distribution service is responsible for routing a distribution from the input queue at the origin DSU to the output queues at the appropriate destination DSUs. The examples illustrated in Figure 10 (derived from Figure 8) show that the output queues may be variously located at the origin DSU, at an adjacent DSU, or at a DSU separated by an intermediate DSU. The processing of routing and directing services shown in Figure 6 for a given distribution begins by accessing an entry from the input queue and ends with the enqueuing of one or more entries either on output queues (if the recipients are local) or on next DSU queues (if the recipients are remote). Discussed next are several cases in which are considered different execution flows of the SNADS model shown in Figure 6.

Forwarding a distribution to another DSU. As shown in Figures 11 and 12, routing a distribution to another DSU may result from a distribution request at an origin DSU or from a distribution that is received from another DSU. In the former case, the routing and directing service first determines the addresses (i.e., destination DSUNs) for the recipients specified in the request. In the latter case—as discussed earlier in this paper in the section on user naming and addressing—routing and directing services may have to replace a destination DSUN in order to redirect the distribution. The subsequent discussion assumes that all destination DSUNs have been successfully determined and reflect the true addresses of the respective recipients.

For each destination DSUN that identifies a remote DSU, routing and directing services must determine the next-DSU queue, the LU name of the next DSU, and the MODE name required for transport services.[19] These parameters are determined by using the destination DSUN and the distribution service level, as shown in Figure 13.

Once an entry is made on a next-DSU queue, the transport service is responsible for transporting the distribution to the next DSU (as discussed later in this paper in the section on synchronous communication between distribution service units). In some cases, a distribution may be routed to the same next DSU for more than one destination DSUN. In these cases, only one copy is sent to the next DSU. That is, only one entry is enqueued to the next-DSU queue, and only one distribution interchange unit is sent to the next DSU. This technique reduces the volume of data being transported in a distribution network. It has been shown[20] that this simple technique is often close to optimal in many typical configurations.

Alternatively, multiple destinations contained in a distribution may result in splitting the original distribution into several distributions to be sent to different DSUs. Figure 14 illustrates both of these cases.

Destination processing may occur as a result of local or remote distributions.

Destination processing. When a distribution arrives at a destination DSU, where one or more recipients

Figure 10 Basic routing cases

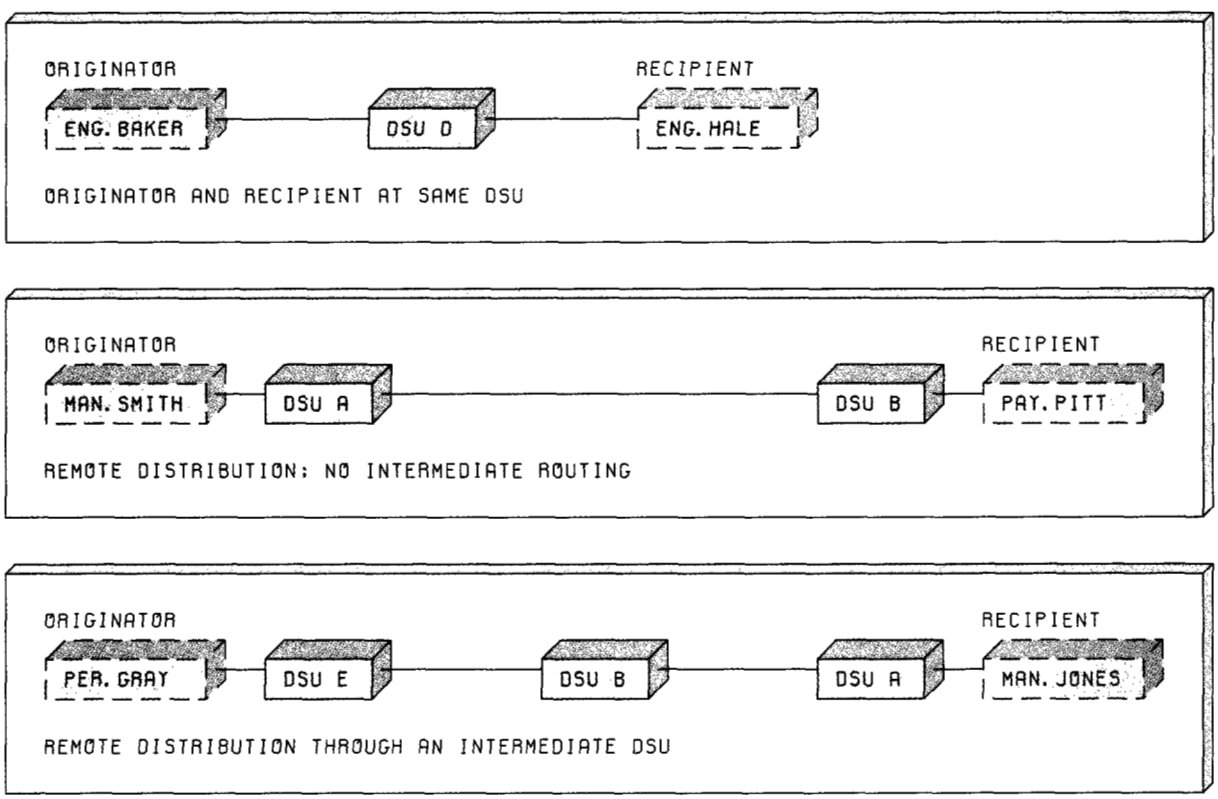
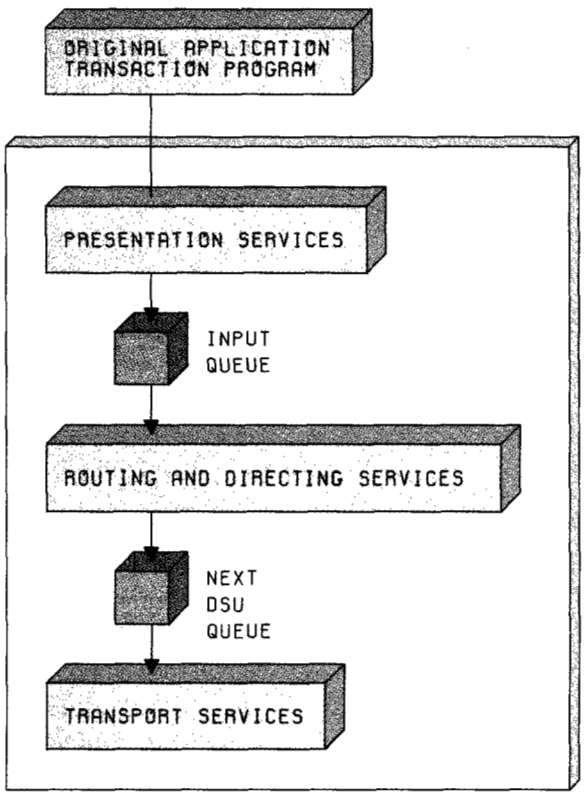


Figure 11 Origin DSU: Routing distribution to remote DSU



are located, routing and directing services enqueue the distribution on the appropriate output queues and (optionally) invoke the destination transaction program to process the request. Destination processing may occur as a result of local or remote distributions.

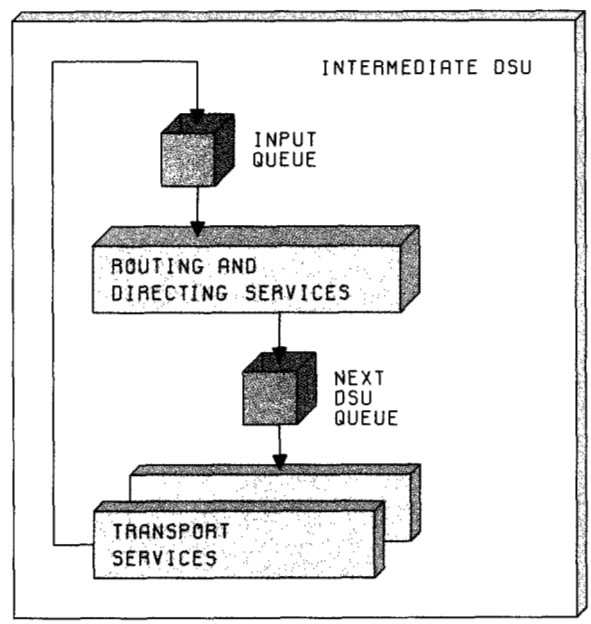
With local distribution, as shown in Figure 15, destination processing occurs at the origin DSU because the recipients of the distribution request are located at the origin DSU. In this case, the origin and destination DSUs are the same, and the functions of transport services are not required.

Remote distributions occur when the originator and recipients are located at different DSUs. In this case, as illustrated in Figure 16, destination processing results from receiving a distribution from an adjacent DSU.

Regardless of whether a distribution request results in local or remote distribution, the distribution function appears the same to users (and to the application transaction programs). This property is sometimes referred to as "local/remote transparency."

The functions of routing and directing services in destination processing are to select the proper output queue and start the proper destination transaction program. The parameters in the distribution used to perform these functions are the destination *Transaction Program Name* (TPN), the distribution service level, and the recipient names. The destination TPN gives originators the ability to select different processing for different distributions addressed to the same recipient(s). The destination DSUs have the responsibility to validate the above parameters (e.g., that a given destination TPN is supported). However, the algorithm and tables that a destination DSU uses to map these parameters to local queues and transaction programs are largely product-specific and may differ among the DSUs in the network. This is because the destination processing is concerned with how distributions are passed to local transactions; it does not affect other DSUs. At one extreme, there may be a separate output queue for each unique combination of destination TPN, DSL, and recipient name. In this case, an entry (containing only one recipient name) is enqueued

Figure 12 Intermediate DSU processing



for each local recipient. At the other extreme, there may be a single queue for each destination TPN, regardless of the DSL or the number of local recipients in the distribution. This, for example, could be a transaction dispatching queue. In this case, a single entry is enqueued that contains all the names of all the local recipients in the distribution. It is the task of the application transaction program to receive the distribution and process each recipient name in the request (e.g., updating mail logs).

Synchronous communication between distribution service units. This section focuses on how two adjacent DSUs communicate synchronously using transport services. Transport services consist of a pair of LU 6.2 transaction programs defined for the synchronous transmission of distributions between DSUs. We discuss first the relationship of SNADS and the facilities provided by LU 6.2 services and synchronous SNA in general. We then describe high-level protocols that have been designed for sending and receiving distributions from one DSU to another. Specifically discussed are instances of starting the sending and receiving transactions, processing next-DSU queues, classes of exceptions that can occur, and the distribution flow-control mechanism.

SNADS and Logical Unit Type 6.2. SNADS was designed assuming LU 6.2 services for interprogram communication. An alternative choice would have been to design a protocol specifically tailored to SNADS requirements. A number of products[21,22] have defined their own interprogram protocols in SNA using Logical Unit Type 0.[23]

LU 6.2 was chosen because it is the IBM SNA interprogram communication protocol and meets the functional requirements of SNADS. It is expected to become widely used for many distributed applica-

Figure 13 *Determining parameters for routing to next DSU*

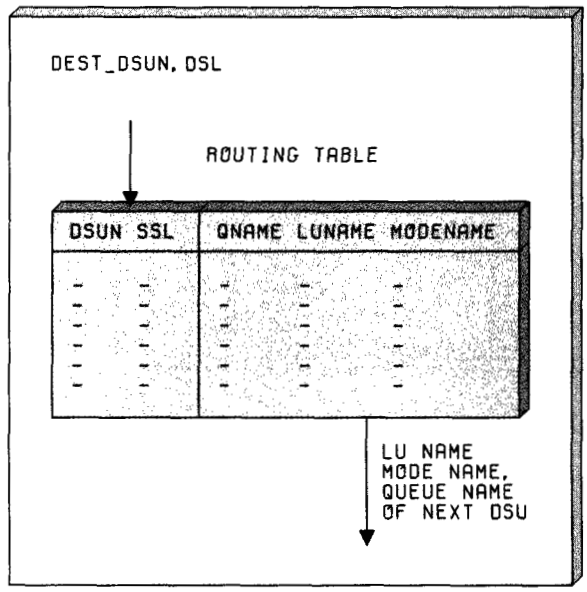
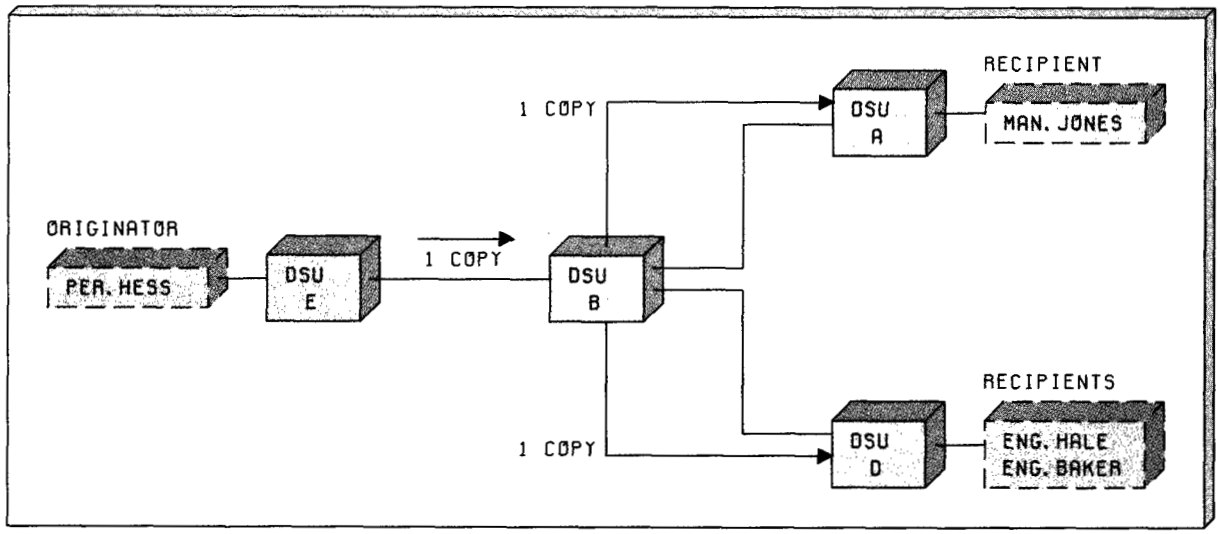


Figure 14 **Distribution fanout**



tions. This common synchronous interprogram protocol provides a basis for implementing SNADS on a wide variety of computers—from intelligent work stations to large processors. SNADS is designed to be able to operate using the minimal LU 6.2 subset (i.e., the LU 6.2 base) that is required in all LU 6.2 implementations.

As discussed in Reference 2, the collective services provided by the community of interconnected logical units can be viewed as a distributed operating system. (See Figure 17.) The activity of exchanging application data (messages) between two LU 6.2 transaction programs is called a *conversation.* The basic functions required by the distribution transaction programs of SNADS are provided by the following LU 6.2 verbs:

- ALLOCATE is issued by a transaction program to establish a conversation with another (partner) transaction program.
- SEND_DATA is used to send application data to a partner transaction program.
- RECEIVE_AND_WAIT is used to receive application data from a partner transaction program. Control is returned to the transaction when the data are available.

Figure 15 Origin and destination DSU with recipients located at origin DSU

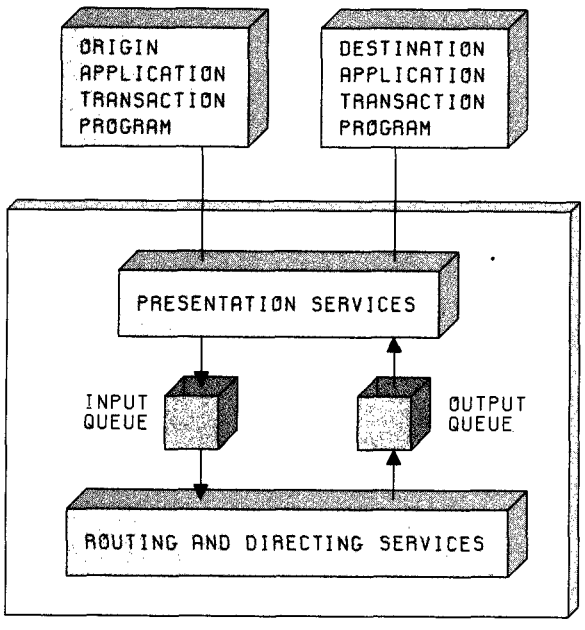
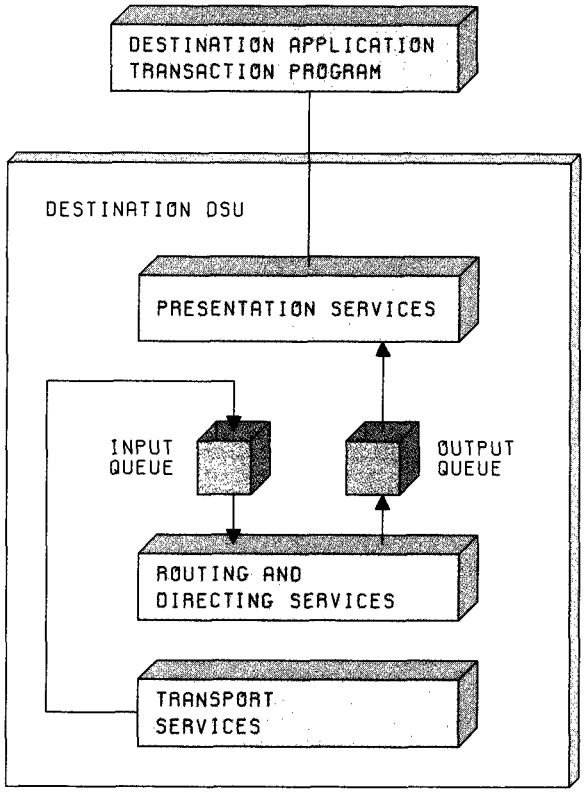


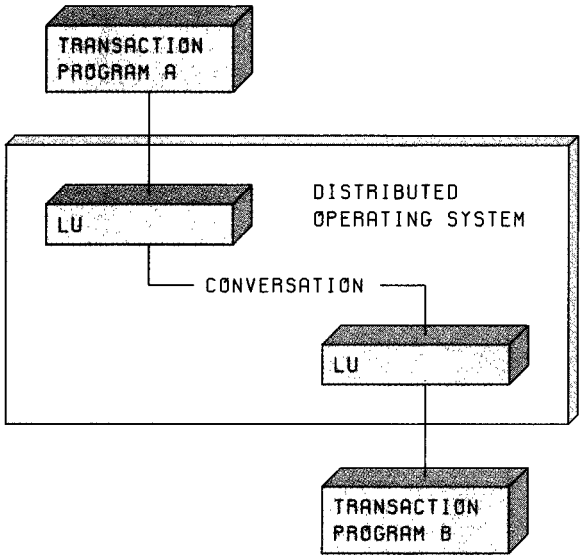
Figure 16 Destination DSU with receiving distribution from a remote DSU



- SEND_ERROR is issued by one transaction program to signal an error condition to a partner transaction program.
- DEALLOCATE is used to terminate a conversation with a partner transaction program.
- CONFIRM ends a message and asks the partner transaction program for assurance that it has taken responsibility for the data. The receiving transaction program may reply with CONFIRMED if it has not detected any errors, or it can issue SEND_ERROR to report exception conditions to the sending transaction program.

Transport services consist of two transaction programs (DISTRIBUTION_SEND, DISTRIBUTION_RECEIVE) that participate in an LU 6.2 conversation to synchronously transfer distributions from one DSU to another. From the perspective of LU 6.2 services, distribution transaction programs are LU 6.2 applications like any other LU 6.2 transaction program. That is, they use the LU 6.2 verbs[3] to allocate conversations, send and receive data, and report

Figure 17 LU 6.2 transaction programs in conversation



errors. Figure 18 shows synchronous communication between two DSUs using LU 6.2 conversations.[24] The pipes illustrate sessions between LU X and LU Y. The dashed lines depict conversations between distribution transaction programs in the respective DSUs (i.e., DSU A and DSU B).

To allocate a conversation with another distribution transaction program, a distribution transaction program must select the name of the LU that contains the partner DSU, the MODE name to be used for selecting the type of SNA session desired, and the name of the distribution transaction program to participate in the conversation. The determination of the LU name and MODE name is described in the previous section on routing and directing services. Distribution transaction program names are architected values and are implicitly known by all DSUs.

In one sense, because SNADS is designed to use the LU 6.2 protocol boundary, distribution networks are intimately linked to synchronous SNA networks. In another sense, however, distribution networks are independent of synchronous SNA networks.

The MODE name insulates DSUs from the details of the synchronous SNA resources. The logic of distribution transaction programs remains unchanged, regardless of the underlying SNA session resources. The MODE name is used by LU 6.2 to determine the SNA session resources and characteristics.[25] It is used to select the SNA class of service and such other session-level attributes as encryption. The class of service, in turn, is used to select an SNA virtual route and the SNA transmission priority. MODE name is also used to determine the number of available parallel sessions (if any).[26]

It should be mentioned that, even though MODE names insulate distribution transaction programs from the details of the LU 6.2 services, the MODE names defined for use by the distribution service are an important distribution network design consideration. If the SNA resources related to a given MODE

Figure 18 Conversation connecting two DSUs

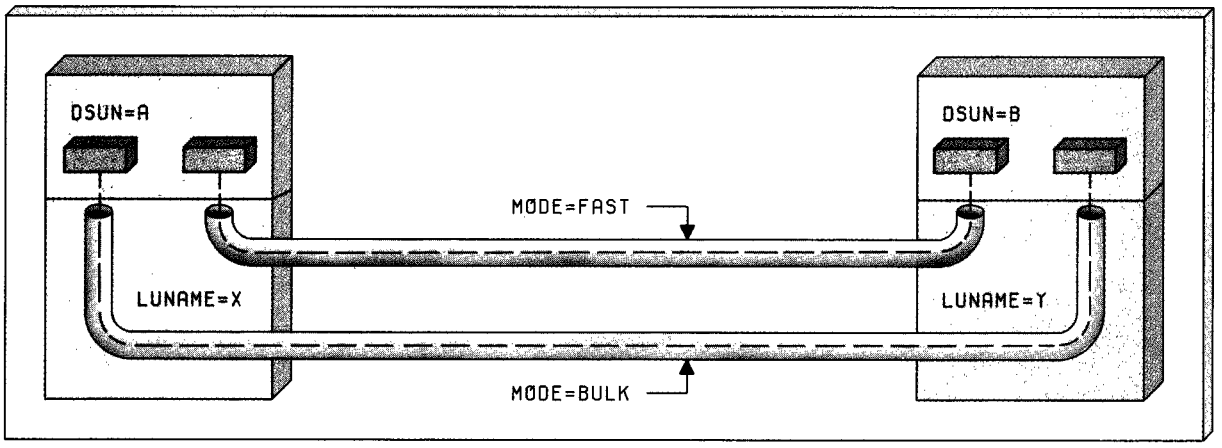
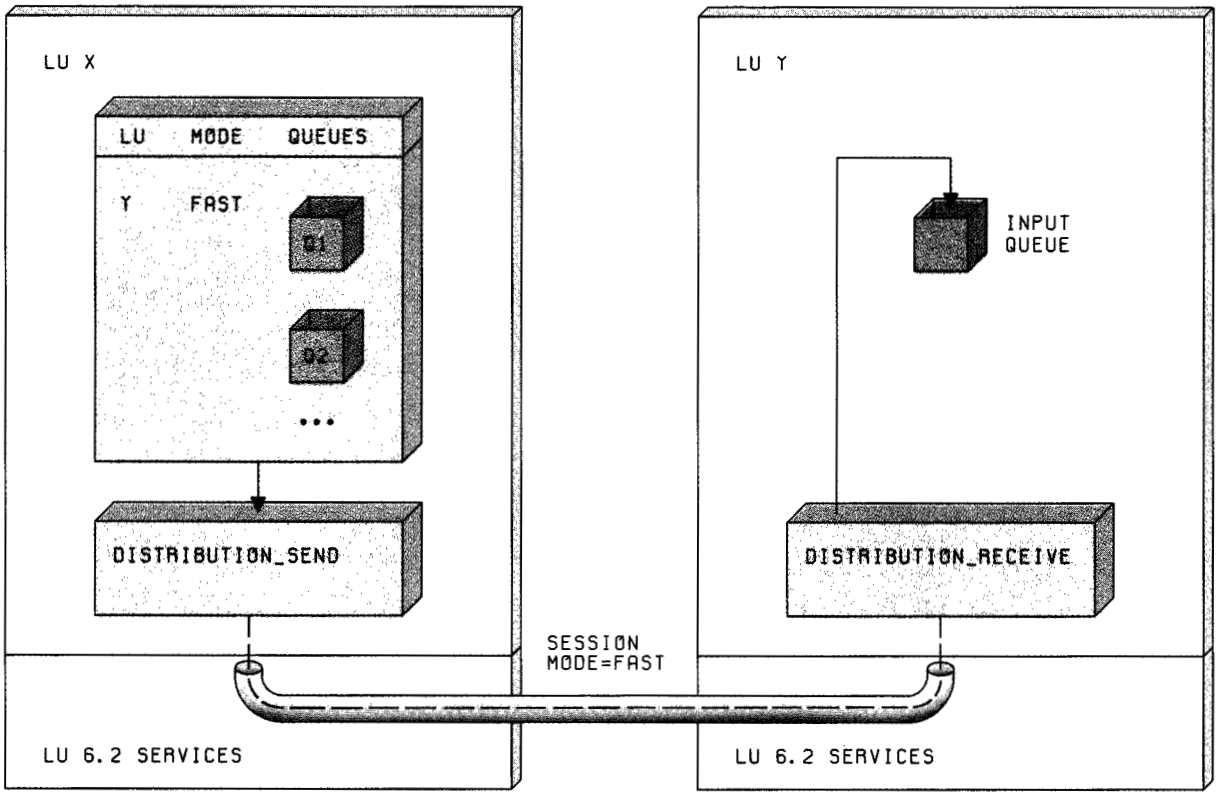


Figure 19 Conversation between DISTRIBUTION_SEND and DISTRIBUTION_RECEIVE



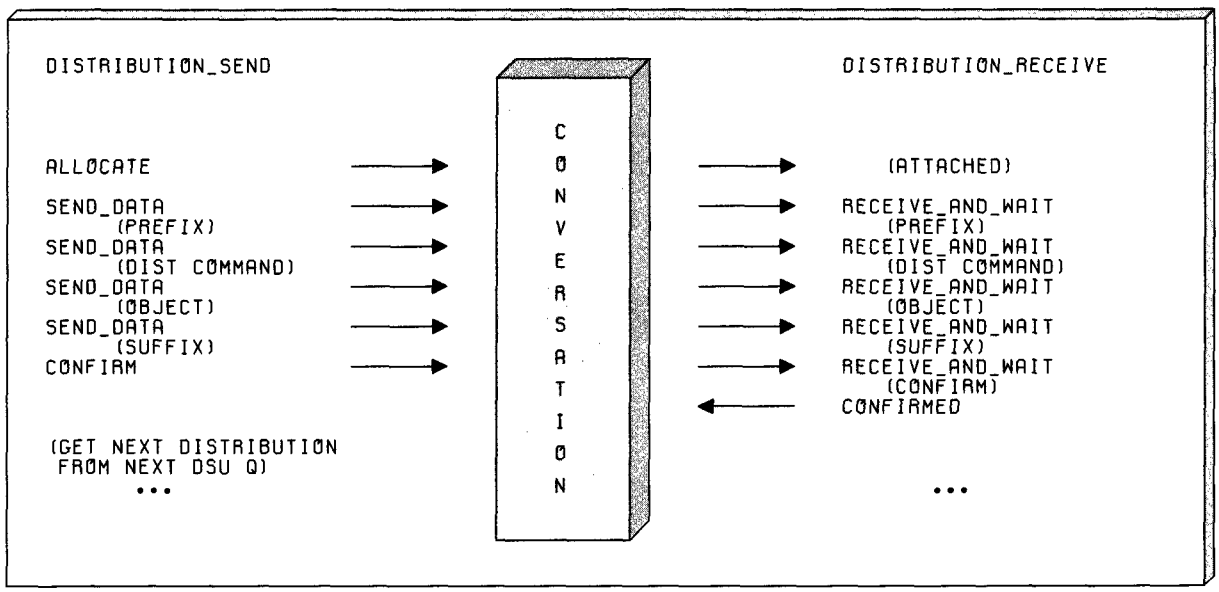
name are mismatched to the associated distribution service levels, unsatisfactory performance results.

SNA Network Interconnection is transparent to SNADS. As described in Reference 27, SNA has a facility for interconnecting two or more SNA networks. With this feature, a session may be established between LUs in different SNA networks. An SNA network interconnection gateway constructs a qualified network name (i.e., NETWORK-ID.LU-NAME) and may translate LU names (if aliasing is necessary). Such translations, however, are transparent at the distribution services layer. Thus, adjacent DSUs may in fact be in separate SNA networks.

DSUs are named independently from LUs. As shown in Figure 18, the DSU names and the respective LU names are different and are selected from different name spaces. LU names are used to establish conversations with an adjacent DSU, but they are never carried in distribution interchange units. This provides additional independence from the synchronous SNA services over and above that provided by LU 6.2. For example, SNADS distributions are not affected by LU name translations that may occur in SNA network interconnections. In addition, this independent naming enables DSUs to be grouped (using routing groups) to satisfy distribution requirements apart from the synchronous SNA topology. The disadvantage of this naming independence is the cost of an additional level of name management.

Initiating data transfer and next-DSU queue processing. Under normal circumstances, routing and directing services at the sending DSU cause an instance of DISTRIBUTION_SEND to be initiated. This may occur for several reasons. The simplest condition is when an a new entry is enqueued to a next-DSU queue. Optionally, some DSUs may initiate DISTRIBUTION_SEND when a certain time of day is reached or when the number of entries in a queue exceed a certain maximum.

Figure 20 Normal operation



As shown in Figure 13, routing and directing services determine the LU name, the MODE name, and the next-DSU queue for a distribution, using the destination DSUN and the distribution service level. It is expected that different MODE names can be used for distributions requiring different distribution service levels. This could be the case, for example, for distributions that vary greatly in capacity or object-size requirements. This would guarantee, for example, that such short messages as notification messages are not serialized behind the transmission of large documents. It may be, however, that the same MODE name is selected for different distribution service levels. For example, two distributions may specify the same capacity and protection levels but differ in their specified priority levels. In this case, the respective distributions may be enqueued on different next-DSU queues that can be serviced on a single conversation, as illustrated in Figure 19. Each next-DSU queue is associated with a unique LU/MODE name. There may be, however, multiple next-DSU queues assigned per LU/MODE name. When an instance of the DISTRIBUTION_SEND is started, the LU/MODE name of the target DSU is passed to it, along with the list of queues that may be serviced once the LU 6.2 conversation has thus been allocated. Priority scheduling is achieved by an ordering of these queues.[28] In the example in Figure 19, Q1 is ordered ahead of Q2. Thus DISTRIBUTION_SEND empties Q1 before proceeding to the next entry in Q2.

To increase throughput, multiple instances of DISTRIBUTION_SEND and DISTRIBUTION_RECEIVE may be started for the same or different LU/MODE names. The precise number depends on such configuration options as the number of parallel sessions allowed for a given LU/MODE name and the maximum number of active transactions allowed within the LU.

Normal distribution processing. Once DISTRIBUTION_SEND is in conversation with DISTRIBUTION_RECEIVE, each distribution is processed as a single unit of work using the LU 6.2 protocol boundary verbs. As shown in Figure 20, DISTRIBUTION_SEND constructs the respective pieces of the distribution interchange unit and sends them to DISTRIBUTION_RECEIVE. The sending of the distribution is complete when an LU 6.2 CONFIRMED is received from DISTRIBUTION_RECEIVE. At this point, the receiving DSU has responsibility for the distribution, and the sending DSU may discard it.

Exception handling and control flow. During the transmission of a distribution, errors may occur. There are two basic classes of errors—*recoverable errors* and *nonrecoverable errors*. Nonrecoverable

errors occur when the receiving DSU determines, for example, that the distribution interchange unit contains an encoding (syntax) error or that the function requested cannot be performed, for example, because the receiving DSU lacks the capabilities specified in the distribution. Alternatively, DISTRIBUTION_SEND may encounter a permanent I/O error. The number of errors detected during the synchronous transfer of a distribution depends on the amount of checking that is done as the distribution interchange unit is being received. For example, routing errors may be detected on-the-fly if the addresses are validated by DISTRIBUTION_RECEIVE. Alternatively, a product may choose to receive the entire distribution and perform the validation in routing and directing services.[29] In any case, when a nonrecoverable error is encountered, the DSU that has responsibility for the distribution discards it and sends a notification message (if requested) to the appropriate user (usually the originator). (This is discussed further in the following section on notification facilities.)

For most applications, there is a requirement to notify users about the state of their distribution requests.

Recoverable errors occur when the resources needed to complete the distribution transfer are temporarily unavailable. The typical example of this is when the receiving DSU runs out of space while storing the distribution object. At a later time, when the space is reclaimed (e.g., due to distributions being forwarded) this distribution may be successfully transferred. To cope with recoverable errors, SNADS provides a hold/release protocol. When a receiving DSU detects a recoverable error, it sets a hold condition, indicating that no more traffic is permitted for the current LU/MODE name. The receiving DSU also sends a negative acknowledge interchange unit (NACK) to DISTRIBUTION_SEND. On receiving the NACK, DISTRIBUTION_SEND holds all the queues (e.g., Q1 and Q2 in Figure 19) that can be serviced on the current LU/MODE name. The releasing of the hold condition can occur in several ways. When the receiving DSU determines that the hold condition has cleared, it may release its hold condition and allocate a conversation with DISTRIBUTION_SEND to release the hold condition at the sending DSU and resume traffic. Alternatively, when some event (e.g., a specified time of day) is reached, the receiving DSU may trigger the sending DSU to try again. That is, the hold condition is reset, a new conversation is established with the next DSU, and transmission is attempted. This may, of course, result in another NACK, if the resources are still unavailable at the receiving DSU. Depending on an algorithm, a sending DSU—after a certain number of retries—may treat the error as nonrecoverable and proceed as described previously. Figures 21 and 22 illustrate the hold-release protocols.

Notification facilities

For most applications, there is a requirement to notify users about the state of their distribution requests. For example, if the distribution service encounters a nonrecoverable error in processing a distribution, the originator (or some user acting on behalf of the originator) should be notified. Similarly, some applications are required to send notifications when certain actions are performed on the received distributions. In office systems, confirmation of delivery is returned (if requested) to the originator when a recipient takes delivery of a distribution.

SNADS provides common facilities for user notification. This facility is used by the distribution service and is also available to any application for reporting status. The decision to provide a common notification facility was motivated by two factors. One is that much of the information necessary for status reporting is the same, regardless of the type of status being reported. All notification messages must contain common correlation information to correlate a notification with the original distribution request. In addition, all notifications must contain the names of the recipients for which the status is being reported. For example, in Figure 14, suppose DSU B encounters a routing error in attempting to route the distribution to DSU A but is successful in routing the distribution to DSU D. The error notification returned to the originator (PER.HESS) must contain the recipient name—MAN.JONES—to identify the recipient affected by the error. The second motivating factor is that having a common notification facility reduces product cost. Application transaction programs must

Figure 21 Recoverable error with HOLD

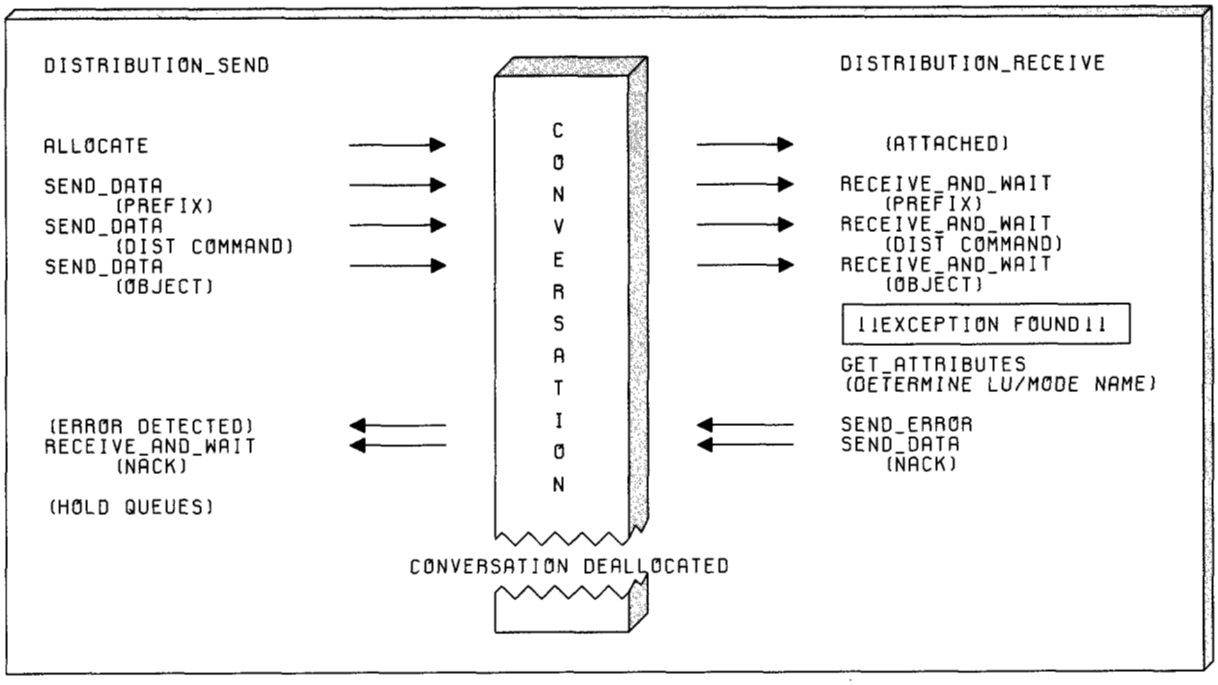
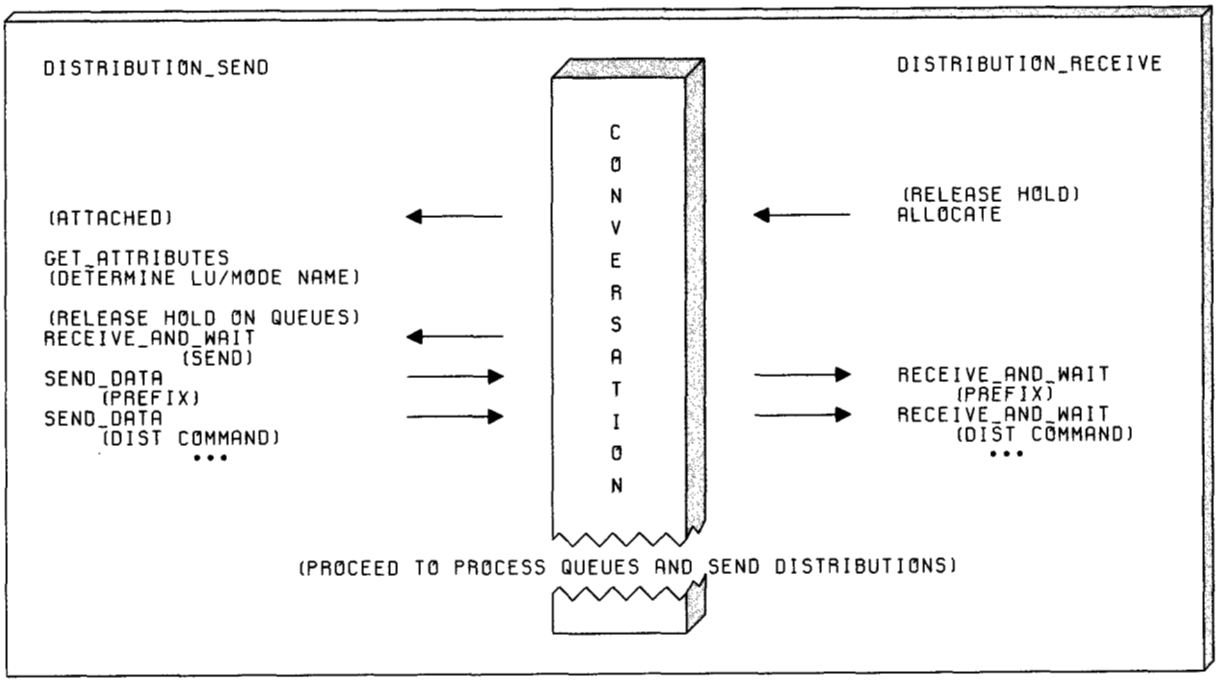


Figure 22 Receiver side release



receive status distributions. Thus, having a general architecture for notifications means that a common implementation can be used to handle notifications of many different types. For example, the status distributions generated by SNADS and DIA office applications can be processed in a uniform way.

For each distribution request, the origin DSU returns (to the origin application transaction program) a Unique Distribution Identifier (UDI). The UDI is carried in all distribution interchange units that result from a distribution request and may be used by applications to correlate notifications with requests.

Optionally, application transaction programs may request notification for distribution requests. If notification is requested (via the DISTRIBUTE_DATA verb), an application transaction program may specify the name of the transaction program and the user to receive notifications for the request. By default, notifications are returned to the originator.

When destination application transaction programs receive distributions (via the RECEIVE_DISTRIBUTION verb), all parameters necessary for notification are returned. The destination application transaction program may then issue (perhaps much later in time) the verb DISTRIBUTE_STATUS to return status. A DISTRIBUTE_STATUS request causes a status distribution to be sent to the specified user and transaction program. Status distributions flow in a distribution interchange unit with extra operands in the distribute command to carry the UDI of the original request, the names of the recipients being reported on, and status information. Status may be reported on behalf of multiple recipients in a single status distribution. Moreover, a different status may be reported for each recipient if desired. Status is specified with a status element that consists of the two fields, status type and status data. The *status type* specifies the category of status (e.g., SNADS or DIA), and the *status data* specify status information with respect to the status type. Each application architecture is assigned one or more status-type values. The application architectures

Figure 23 DIA using SNADS to report confirmation of delivery

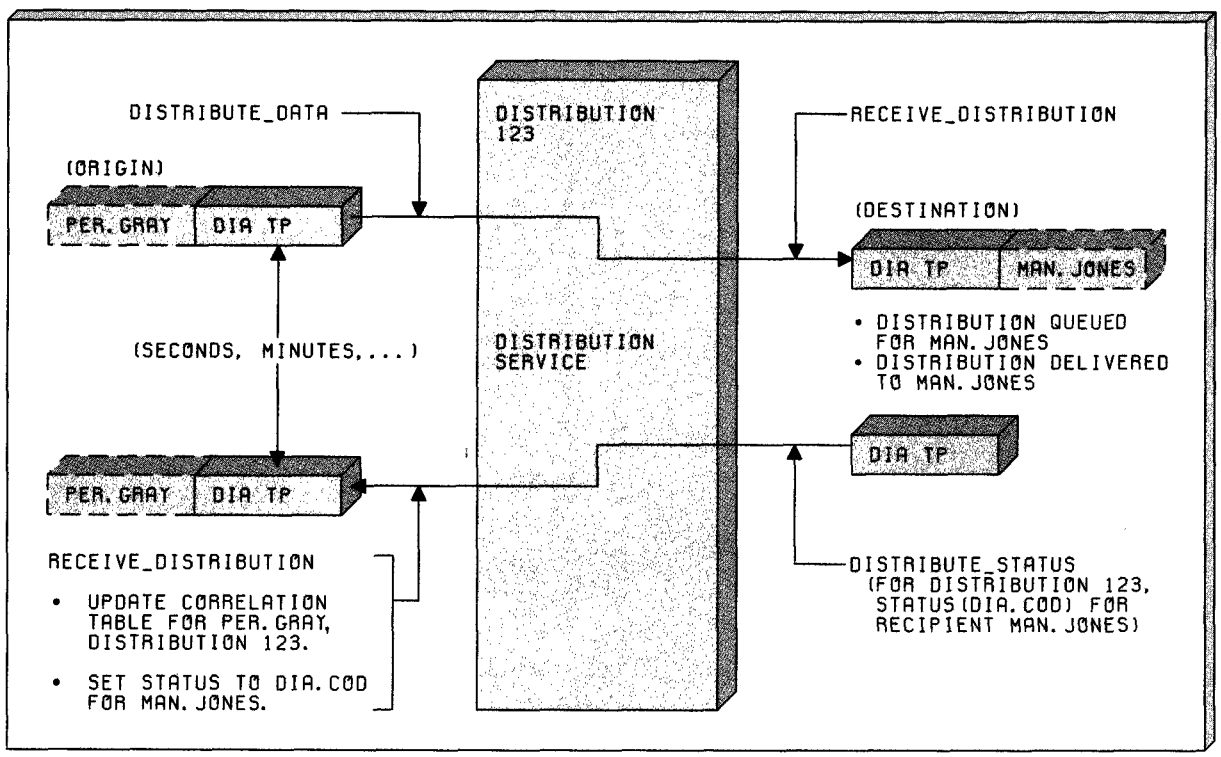
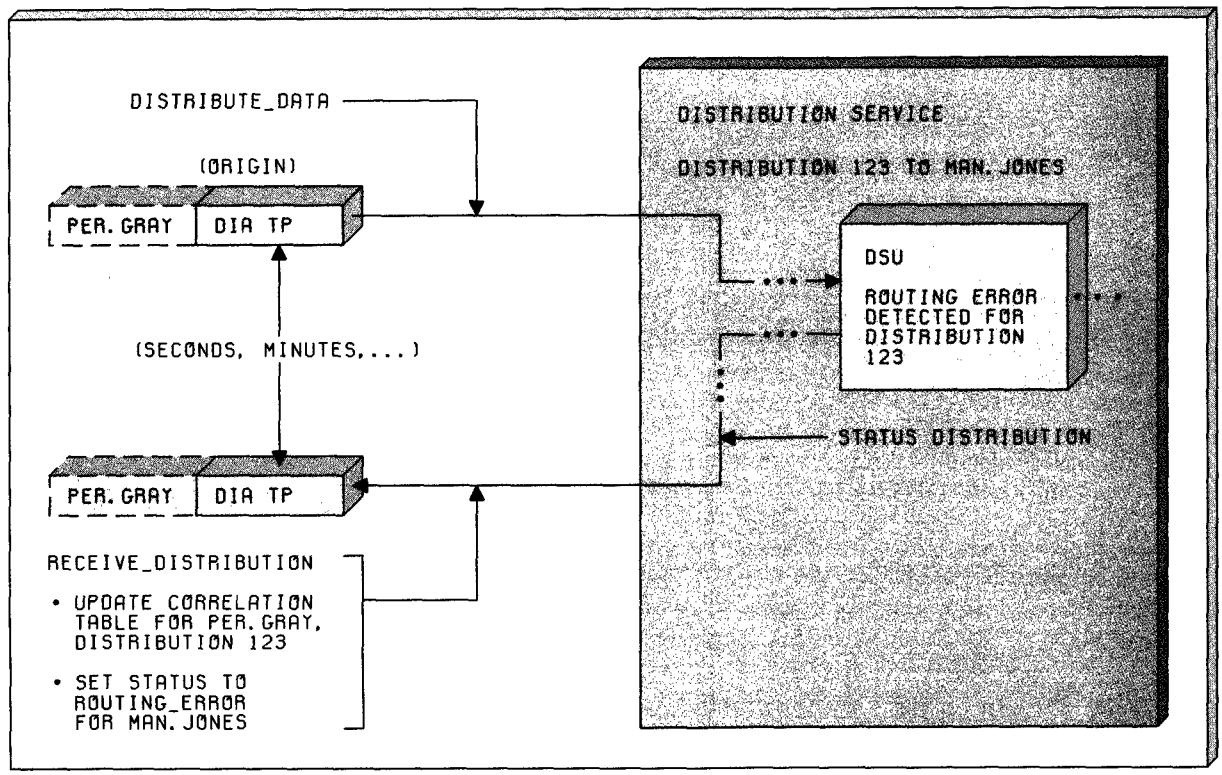


Figure 24 SNADS reports routing error to DIA transaction program



are then free to define the contents of their status data. Figure 23 shows how the SNADS notification facility can be used by a DIA application transaction program to report confirmation of delivery. The number 123 denotes a unique identifier assigned by the distribution service (i.e., the origin DSU) for the distribution. The number 123, coupled with the originator's name (PER.GRAY) forms a network-unique identifier for the distribution.

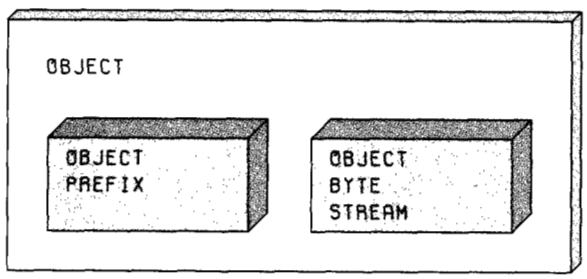
Figure 24 illustrates an SNADS-detected error being reported to the originator. This example is similar to Figure 23, except that a DSU in the distribution network has encountered a routing error while processing the distribution. The DSU builds a status distribution with a status type of SNADS and sets a condition code (in the status data) indicating a routing error. To assist in problem determination, the detecting DSU logs all errors and related control information and includes the DSUN of the detecting DSU in status distribution.

Servers

Servers are used by a DSU to retrieve and store the distribution objects. The server protocol boundary consists of a set of verbs for server initiation and termination, for reading and writing the object byte stream, and for controlling shared access to the stored object(s).

As previously stated, SNADS must be able to handle efficiently a wide range of object sizes. In the past, most asynchronous distribution systems have required that the distribution object be moved from the user's space to a temporary store such as a spool file. The requiring of a temporary store is attractive from the standpoint of simplicity in that a DSU can use a common server for all object accesses. This avoids problems associated with direct access to the user's space, as in the sharing of access between applications and the DSU and in the mapping of the object byte stream to the form required by the

Figure 25 Components of an OBJECT



application. Unfortunately, for very large objects it is undesirable and sometimes infeasible to require that the object be moved between the user's storage (e.g., a data set or document library) and a temporary store at the origin and destination DSUs.

SNADS requires direct access to objects. At the origin DSU, it must be possible to retrieve the object byte stream directly from the user's (application's) space. Similarly, at the destination DSU, it must be possible to write the object byte stream directly into the user's (application's) space.

Of course, different applications may distribute a variety of different object types (e.g., documents or facsimile) and may require different data management routines for retrieving and storing data. Thus, users must be able to name the server to be used to access objects at the origin and destination DSUs. Parameters are defined on the distribution verbs for specifying the origin and destination server names. As illustrated in Figure 25, the OBJECT depicted in Figure 5 consists of an *object prefix* and the *object byte stream*.

The object prefix contains the name of the server (and optionally a parameter string) to be used to store the object at the destination DSU. The object byte stream is the byte stream that is delivered to the origin DSU by the origin server and delivered to the destination server by the destination DSU. The object byte stream may be segmented[30] and can be any length, subject to storage capacity limitations.

The relationships of servers, DSUs, and application transaction programs are illustrated in Figure 26. Here the server processes are defined as being outside the distribution service. A set of verbs that make up the server protocol boundary is defined by the SNADS architecture to describe the functions required by DSUs in accessing objects. These functions are general and apply regardless of the type of server being used.

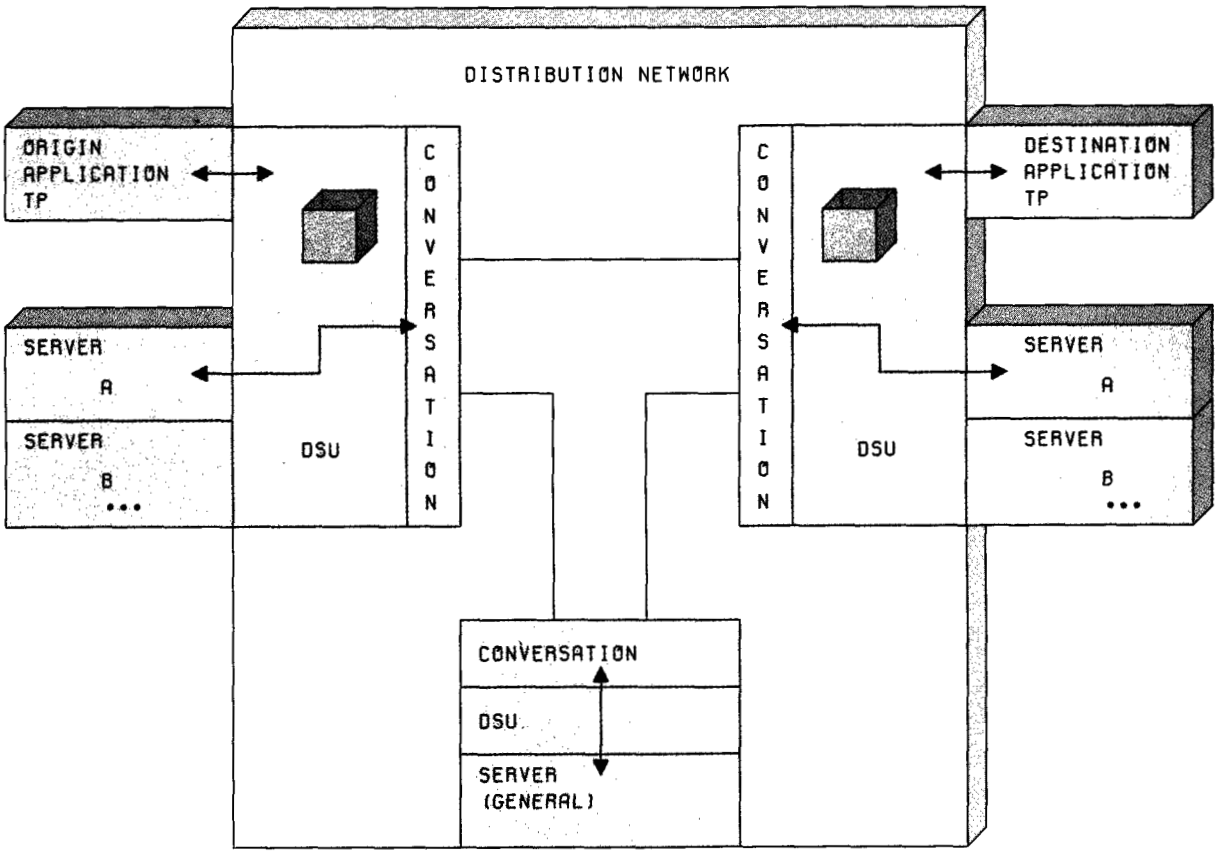
Figure 26 shows that there are two classes of application-specific programs defined in the SNA Distribution Services architecture: (1) origin and destination transaction programs that send and receive distributions, and (2) servers that access the distribution objects. Servers are invoked by a DSU (on-the-fly) as an object is being sent or received on an LU 6.2 conversation. From the viewpoint of SNADS, servers act as a source or sink for the object byte stream. SNADS uses the server name to determine which server to call, but otherwise SNADS has no information on the semantics of the object byte stream or any associated server-specific processing. There are two classes of servers defined in SNADS—*specific servers* and *general servers*.

Any server may serve as a general server if it can store and retrieve a byte-perfect copy of the object.

The function of a specific server is to provide the mapping between the object byte stream and the appropriate application-specific form. Specific servers typically have information regarding the meaning of the byte stream. The potential functions of a specific server include the following:

- *Encoding and decoding the data contents.* Many byte streams carried by SNADS are architected data streams. The server must encode and decode the application data stream (e.g., DIA document units).
- *Server-specific profile processing.* Profiles such as the DIA Interchange Document Profile (IDP) must be handled by DIA servers. The distribution service has no information on such entities. This means that servers may have to read or write catalogue entries maintained for different data sets and libraries.

Figure 26 Distribution network and file servers



- *Server-specific transformations.* Some servers may, based on information passed in the object prefix or in application-specific profiles, choose to transform the data stream (e.g., compression/decompression) as part of the mapping to and from the data store.
- *Interfacing with local data management components.* The server must determine—using the server name, server parameters, and server profile information—which local access method or library service to invoke to read and write the data stream.

General servers are required by intermediate DSUs. The function of general servers is to write and read (when the distribution is forwarded) a byte-perfect copy of the distribution object(s). A general server is not sensitive to the semantics of the distribution object (object prefix or object byte stream). Thus the inherent difference between specific and general servers is the extent to which information on the byte stream is required. Any server may serve as a general server if it can store and retrieve a byte-perfect copy of the object. As illustrated in Figure 26, the general server is considered to be part of the distribution service because it is required by all intermediate DSUs.

A number of complexities arise when the function of direct access to objects is provided. One such example is the synchronization required between applications and a DSU. When an application submits a distribution request, it must be possible to guarantee integrity to ensure that the object has not been modified before the DSU has relinquished responsibility for the distribution. When more than one copy of an object must be forwarded, the object cannot be deleted until all copies have been transmitted. The server verbs specify locking protocols to handle object synchronization requirements.

Concluding remarks

This paper has presented the SNA Distribution Services (SNADS) architecture, that is designed for asynchronous data distribution for SNA applications running in an LU 6.2 environment. The requirement for a general distribution facility in today's rapidly expanding computer communications technology is evidenced by recent standards activities.[31-33]

The general design objectives and requirements have been briefly discussed, the basic terminology introduced, and a technical overview of the IBM SNA Distribution Services has been presented. The remainder of the paper has focused, in some depth, on the primary concepts of naming and addressing, distribution service levels, distribution routing, synchronous communication among distribution service units, user notification facilities, and object servers.

Although the generality of the architecture has been emphasized, it should be stressed that the predominate application of SNADS at the present time is that of office systems. Toward that end, SNADS has been designed to be compatible with the Documentation Interchange Architecture. The initial implementations of SNADS are provided by DISOSS Version 3 Release 2 and the IBM 5520 Release 5.

Acknowledgments

The development of the SNA Distribution Services architecture has been a joint effort. Many architects and product representatives have contributed to the requirements and technical design. We are indebted to E. H. Sussenguth and J. P. Gray for their vision regarding the need for a general asynchronous data distribution facility for SNA applications. Among the architects who have contributed to SNA Distribution Services, we wish to acknowledge J. C. Ashfield, J. M. Baker, P. F. Chimento, J. P. Gray, J. C. Knott, L. T. O'Connor, S. Shukuya, and P. N. Turcu. We also thank L. E. Area and K. Knight from the IBM 5520, S. D. Hale, J. R. Hind, and L. F. Morrison of the IBM Distributed Office Systems Support (DISOSS) product, and R. F. Brockish of the Document Interchange Architecture. Finally, we thank the management team, including R. F. Steen, J. C. Broughton, T. B. McNeill, M. L. Hess, D. A. Haile, E. W. Cornish, E. R. Roth, and P. B. Hill.

Cited references and notes

1. Implementations of SNADS will be announced on a product-by-product basis.
2. J. P. Gray, P. J. Hansen, P. Homan, M. A. Lerner, and M. Pozefsky, "Advanced program-to-program communication in SNA," *IBM Systems Journal* **22,** No. 4, 298–318 (1983, this issue).
3. *Systems Network Architecture: Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084, IBM Corporation; available through IBM branch offices.
4. A *distribution transaction program* is any transaction program that is defined as part of the SNADS architecture to provide required functions of the distribution service.
5. The term *distribution* refers to an entity resulting from a distribution request that is transported by SNADS from an origin to one or more destinations.
6. The formal term *Distribution User Name* (DUN) is defined in the SNADS architecture to designate a distribution user. For convenience in this paper, the simple terms "SAF user" or "user" are used unless formality is required.
7. The term *protocol boundary* refers to the architectural definition only. Products implementing the architecture may not define their interfaces in the same way or support all the functions described.
8. M. R. DeSousa, "Electronic information interchange in an office environment," *IBM Systems Journal* **20,** No. 1, 4–22 (1981).
9. T. Schick and R. F. Brockish, "The Document Interchange Architecture: A member of a family of architectures in the SNA environment," *IBM Systems Journal* **21,** No. 2, 220–244 (1982).
10. B. C. Housel, "On the design and formal description of messages in distributed architectures," *Proceedings of the International Conference on Computer Communications (ICCC)*, London (September 1982), pp. 627–633.
11. The acknowledge interchange unit is synonymous with the Document Interchange Architecture (DIA) acknowledge document interchange unit, as defined by DIA. SNADS uses this interchange unit for reporting exceptions to the sending DSU only. In SNADS, positive acknowledgment is done using the LU 6.2 verb CONFIRMED.
12. This model is defined for the purpose of architectural description; other models may be equally valid. Products may structure their implementations differently.
13. The presentation services defined for SNADS may be viewed as a procedure of SNA presentation services depicted in Figure 4. Upon recognizing a distribution services verb, SNA presentation services calls distribution presentation services and passes to it the verb and its operands.
14. In a mixed network where some DSUs support RGN and other DSUs do not, certain naming restrictions are necessary. These alternatives are not discussed here.
15. In this paper, the period is used to depict levels of qualification in naming (for example, a.b). Periods are not required to encode names in the SNADS architecture, because each part is encoded using self-defining constructs.
16. J. P. Gray and T. B. McNeill, "SNA multiple-system networking," *IBM Systems Journal* **18,** No. 2, 263–297 (1979).
17. The architecture is an open-ended one in that additional capabilities may be added as required.
18. By *integrity* we mean the ability of SNADS to verify the delivery of a distribution. The degree of integrity required may vary among different applications. At one extreme, the

distribution of sensitive material requires extremely high integrity. On the other hand, the distribution of general announcements for a meeting may not require any integrity. Also, the value of some distributions depreciates over time.

19. In some systems (e.g., IMS, TCAM), the session resource can be implicitly associated with a specific (transaction) queue. In the SNADS model, these are treated separately to show clearly the mappings between the SNADS names and the resources needed for LU 6.2.
20. K. Bharath-Kumar and J. M. Jaffe, "Routing to multiple destinations in computer networks," *IEEE Transactions on Communications* **COM-31,** No. 3, 343–351 (1983).
21. *Customer Information Control System/Virtual Storage (CICS/VS), Version 1 Release 6: General Information,* GC33-0155, IBM Corporation; available through IBM branch offices.
22. *Network Job Entry for JES2,* GC23-0100, IBM Corporation; available through IBM branch offices.
23. Logical unit type 0 (LU 0) permits applications to use the basic SNA functions in a product-specific manner. The use of brackets, the choice of half-duplex or full-duplex transmission, protocols for backout and recovery, and many other functions are left to product choice.
24. Henceforth, the term *conversation* is used to imply an LU 6.2 conversation. In contrasting conversations with sessions, a CONVERSATION provides a communication path between two transaction programs, whereas a SESSION defines a path between two LUs. A conversation requires a session. A session, however, may be serially allocated to many conversations over time.
25. See References 16 and 26 for detailed discussions regarding SNA virtual routes, class of service, and other SNA concepts and facilities of synchronism.
26. *Systems Network Architecture: Concepts and Products,* GC30-3072-0, IBM Corporation; available through IBM branch offices.
27. J. H. Benjamin, M. L. Hess, R. A. Weingarten, and W. R. Wheeler, "Interconnecting SNA networks," *IBM Systems Journal* **22,** No. 4, 344–366 (1983, this issue).
28. The SNADS model shows multiple ordered queues. An implementation might have one queue in which entries are enqueued/dequeued in priority order.
29. Routing and directing services run asynchronously with respect to DISTRIBUTION_RECEIVE and, therefore, with the conversation. There are design tradeoffs to be considered in determining where error checking and address validation are to be performed. Extensive processing in DISTRIBUTION_RECEIVE may underutilize the session. On the other hand, if errors can be detected as the distribution is being received on the conversation, the wasted cost of storing large objects is avoided.
30. Distribution objects are segmented using the segmentation technique defined for DIA Document Units.
31. There has been increasing interest in asynchronous data distribution technology. Recent standards work[32,33] has resulted in a proposed standard—called Message Handling Facility (MHF)—for Computer-Based Message Systems (CBMS). At a high level, there are obvious similarities between the SNADS model and the MHF model. For example, the MHF User Agents are analogous to application transaction programs, and the MHF Message Transfer Agents are similar to the DSUs.
32. T. H. Myer, "Global messaging—issues and approaches," *Journal of Telecommunications Networks* **1,** No. 2, 173–187 (1982).
33. CCITT Study Group VII, *Message Handling Systems Recommendations,* International Telephone and Telegraph Consultative Committee, March 1983. (This work consists of a number of recommendations, X.MHS 0 through X.MHS 7, that address the various aspects of message handling.)

Reprint Order No. G321-5198.

Barron C. Housel *IBM Communication Products Division, P.O. Box 12275, Research Triangle Park, North Carolina 27709.* Dr. Housel attended the University of Oklahoma, Norman, Oklahoma, where he received a B.S. degree in mechanical engineering in 1963 and an M.S. degree in engineering science in 1965. He joined the IBM Corporation in 1965. Dr. Housel received an M.S. degree in computer science from Stanford University, Stanford, California, in 1968 and a Ph.D. in computer science from Purdue University, Lafayette, Indiana, in 1973. From 1973 to 1979, Dr. Housel was a member of the IBM Research Division, where he did research in data base technology. During that period he was a visiting professor in computer science at Purdue University. Since 1979, Dr. Housel has been a senior engineer in the architecture and telecommunications department at the IBM laboratory in Raleigh, North Carolina.

Carol J. Scopinich *IBM Communication Products Division, P.O. Box 12275, Research Triangle Park, North Carolina 27709.* Ms. Scopinich joined IBM in the Federal Systems Division in 1968 in Owego, New York. There she was involved in the design and development of software to support on-board military flight systems. In 1970, she moved to Morris Plains, New Jersey, to work on research and development for the Safeguard project. After moving to Raleigh, North Carolina, in 1974, she did design and development for the control program of the IBM 3650 Retail Store System. From 1976 to 1978, she was responsible for designing portions of the Network Control Program. During the past three years Ms. Scopinich has participated in the design of Systems Network Architecture. She has been on brief assignments to England and Italy. Ms. Scopinich received a B.A. from Mansfield State College, Mansfield, Pennsylvania, in 1967 and an M.S. in mathematics from Clarkson College of Technology, Potsdam, New York, in 1968.