System-independent file management and distribution services

by J. C. Ashfield
D. B. Cybrynski

Applications universally require files to move from one location to another. Although different applications may use files differently, many of the files are of the same type. The identifying, fetching, moving, and storing functions are the same for all applications and can be most efficiently provided by a common process. Various applications can invoke the common process, which performs the required operations independently and notifies the appropriate applications when they are completed. In Systems Network Architecture (SNA) networks, the common process is an SNA/Distribution Services (SNA/DS) server defined by SNA/File Services (SNA/FS). The invoking applications are SNA/DS agents of various types. This paper describes the role of the agent in invoking the file transfer and the role of the SNA/FS server in fetching and storing the file. It also describes the SNA/FS architecture for uniquely naming files and data objects. One example of an SNA/DS agent that uses the SNA/FS server is the change management category of SNA/Management Services, described in another paper in this issue.

When an organization connects two or more of its computing systems together by means of communication links, the first use of the connection is typically to move copies of data files or programs between the systems. For small networks of homogeneous systems, simple point-to-point transfer programs administered by operators at both ends are adequate. The successful operation of a variety of file transfer products attests to the practicality of this approach.

For large networks of heterogeneous systems with heavy volumes of file movements, the situation is different. The techniques that work adequately in small homogeneous networks give rise to difficulties when attempts are made to apply them to large networks. Systems Network Architecture/File Services (SNA/FS),[1] has been designed to overcome the difficulties:

1. Limitations of point-to-point—Most file transfer products presume a point-to-point operation, permitting operators at both ends to be involved in the file movements. The operators can plan the transfer work and resolve problems by a telephone call.

   In large networks, file transfer operations may involve more than two locations. A requester at one node might wish to move files from a second node to a third node, and perhaps to a fourth and fifth node. Sometimes copies of one file are sent to hundreds of nodes. It would be impractical to involve operators at every location and to attempt to resolve problems by telephone.

Increasingly, large networks include personal systems and small departmental systems. These systems are often unavailable simply because they are turned off. It would be impractical to wait until all the nodes required for a particular file transfer happen to be available.

SNA/FS file transfers can involve many nodes in various roles. SNA/FS defines extensive exception-detection and reporting mechanisms to facilitate unattended and centralized operations for nodes in any role. It uses a store-and-forward communication facility to transport requests, files, and reports between locations so as to obviate the need for concurrent availability.

2. Problems due to local naming—Typically, data processing operating systems were developed on the assumption that each system would operate independently, with total control over all of its resources. The effect of this assumption particularly impacts the techniques used to identify the files that are moved around the network. Traditionally, each system controlled the names of its files and would ensure that the names were unique. For example, at system A a name such as FILE1 would refer to one and only one data file. At system B, however, the name FILE1 could refer to a completely different data file. This approach is termed local, or location-dependent, naming.

   Anyone wishing to send a file to a particular system, called the *target* system, would be required to identify the file according to whatever resource-naming rules applied at that system. In other words, no matter what name might have been used to identify the file at the location it is to be sent from, called the *source*, the name used at the target would have to conform to the rules there. For example, if a user wished to send the file known as FILE1 from system A to system B, the user would have to ensure that another FILE1 did not already exist on system B. If it did, the user would have to assign the file a new name that did not conflict with any of the existing names at the target. Typically, the names at the target system would conform to a naming convention established by the people responsible for operating that system. Therefore, the people at the source system would have to be aware of both the already assigned names at the target and the conventions used for assigning new names there.

   When copies of files have been sent to other systems, there is often a need to maintain records of what files are at which locations or, alternatively, to enquire of a location what files reside there. With purely local naming, most files would have a different name at every system. For networks of even modest size, purely local naming results in unacceptable complexity.

   The alternative to local naming is global naming. SNA/FS formally defines a naming technique that supports global naming.

3. Naming incompatibilities with heterogeneous systems—A further naming complication arises when networks contain a mix of system types. Each system type has its own rules for expressing file names. Usually, the names are split into several *tokens*. The number and lengths of these tokens, the character sets allowed, method of delimiting tokens, and overall maximum lengths, all differ from system type to system type. A *lowest common denominator* set of rules could produce names that would be legal in all systems, but such names would be too limited to support global naming.

   SNA/FS defines rules for encoding its global names that are independent of all system-specific naming rules and comprehensive enough to meet the requirements for enterprise-wide and industry-wide global names.

4. Informal and system-specific version control—When files are updated, it becomes necessary to distinguish between old and new versions without losing the basic identity of the file. There are various system-defined mechanisms for this, such as the generation data groups in Multiple Virtual Storage (MVS). Alternatively, some applications implement their own private method of identifying versions. When objects are moved from one system type to another and from one application area to another, the incompatibilities between the various forms of version control are a problem.

   SNA/FS defines a mechanism called *partial naming* by which each application can specify which parts of the global name are to be used for version control; these parts need not be precisely specified by the user.

5. System-specific file classification—Many of the system-specific naming conventions are designed both to identify an individual file and to indicate particular file attributes. For example, in one system the name FILE1 SCRIPT A identifies FILE1 and indicates that it is appropriate input for pro-

grams that process SCRIPT. In another system DIR5\PROG2.EXE identifies PROG2 in directory DIR5, and the EXE indicates that it is executable. The tokens SCRIPT and EXE classify the file. Processes that operate on only certain classes of files test these tokens and depend upon their presence and correctness; other processes ignore them. Their significance depends upon the intention of the process. For example, a copy utility might handle files of any class, whereas a loader might refuse to load any files that are not of the class EXE because the intention of the loader is to initiate execution. Unfortunately, different systems use different approaches to classify files. The variety of systems in a mixed network further complicates the problem, because what is executable in one system is probably not executable in another, although it should be able to be stored there.

In SNA/FS, file classifications are explicitly specified in canonical classification codes that are independent of the global name. In addition, SNA/FS defines an intention code so that the file classification and intended use can be related to the capabilities of the target.

Problems not addressed by SNA/FS. The following three items are areas in which problems occur that are not addressed by SNA/FS:

1. Record-level processing—SNA/FS is designed to work with whole files. It does not identify individual records within a file. Accessing individual records within a file on a remote system is best done using implementations of Distributed Data Management (DDM) architecture.[2]
2. Contents conversion—Although it would be convenient for the transport facility to resolve minor incompatibilities between files from different systems, such as ASCII versus EBCDIC codepoints for text, SNA/FS does not attempt to do so. Although ASCII versus EBCDIC is an example of a characteristic that can apply to the file as a whole, it is an exception. Most differences between files are not properties of the file as a whole. Usually the differences are at the level of fields within a record; some fields are characters, others are various sorts of arithmetic variables. Detailed definitions of each field are needed to convert such files. SNA/FS neither identifies records nor defines fields within records.
3. Communications—SNA/FS defines the fetching and storing of files at their origin and destination. The queuing, scheduling, and managing of the communications is performed by SNA/Distribution Services (SNA/DS).[3,4] The actual transfer of data is performed by Logical Unit Type 6.2 (LU 6.2)[5,6] and the lower layers of SNA.

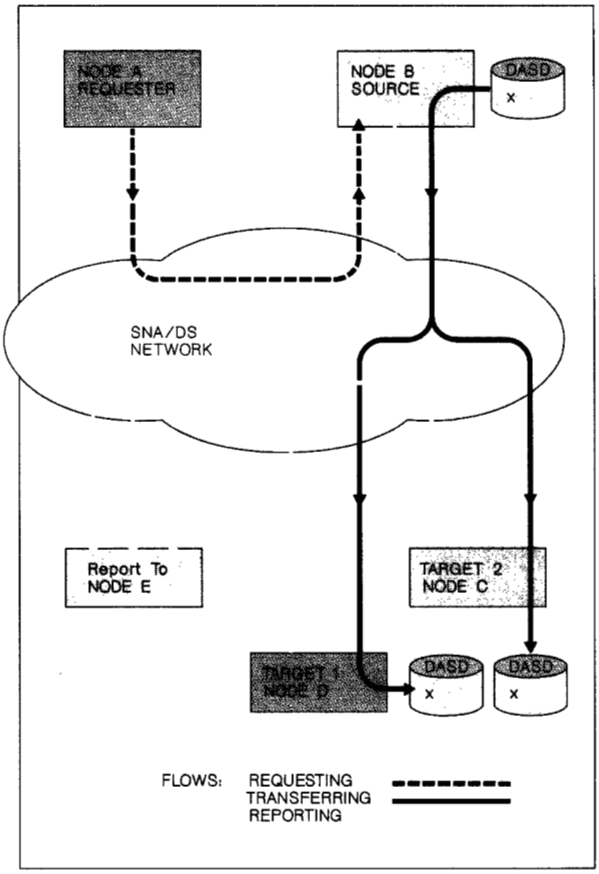
Locations and roles. SNA/FS has a rich concept of locations and the various roles that each location can play. This characteristic contrasts strongly with single-system image architectures which provide transparency. In other words, they completely shield the user from having to be aware of locations. DDM[2] is designed to operate with that kind of transparency. SNA/FS would have a limited purpose in such an environment. Not only are SNA/FS requesters usually aware of other locations, they know when those locations need to have files transferred to them.

SNA/FS is designed to work with whole files.

SNA/FS location concepts are also richer than the pair-oriented architectures used to describe point-to-point/terminal-to-host connections, for example, LU 6.2 and LU 2. Point-to-point situations have an advantage in that problems of capabilities and compatibilities are readily resolved by immediate negotiation. SNA/FS cannot presume a point-to-point, concurrent-availability situation. Therefore, the most likely problems must be anticipated, and their preferred solution must be indicated or implied in the request.

SNA/FS defines four roles. All of them can be involved in a single request, and there can be multiple nodes in the target role. For example, as shown in Figure 1, a user at node A might request node B to send a file to nodes C and D. SNA/FS identifies node A as the requester location, node B as the source location (meaning the location that contains a copy of the file to be distributed), and nodes C and D as target locations (meaning the locations to which a copy of the distributed file is sent).

Figure 1 SNA/FS roles and locations



SNA/FS also has the notion of a report-to location. SNA/FS reports can be sent to whatever location the requester chooses. Figure 1 depicts a typical unit of work for SNA/FS. The requester is located at node A and the source at node B. C and D are two target nodes, and E is a report-to node. The cloud-like shape represents a general-purpose distributed system (DS) network, consisting of an indefinite number of intermediate DS nodes, none of which needs to have any SNA/FS capability. The various lines through the cloud depict the request, transfer, and report flows required to perform an SNA/FS operation.

Not all SNA/FS requests involve this many locations. Sometimes there is only one target, and one location can serve multiple roles. In the case of a file fetch, the requester and target are the same location. In the point-to-point transfer case, the requester and source are the same. Often, the report-to and the requester location are also the same.

SNA/FS global naming

SNA/FS introduces a formally-defined global name for files and data objects. In distributed networks, those files that are of interest beyond their local system are assigned a name that is unique within the wider area of interest. The term *global* is used to distinguish this type of name from the traditional local name.

Global naming itself is not a new concept. The notion of global LU names was introduced to SNA some years ago. The global naming of files has been practiced within organizations on an informal basis since the impracticality of local-only naming became apparent. SNA/FS formally defines a structure for a global name and extends the area within which a global name must be unique to encompass all organizations using SNA. In other words, SNA/FS defines a cross-enterprise global name. It also formally defines an encoding of the name token string that is independent of operating system specifics.

The simplest technique for assigning global names is to concatenate the location name (assuming that it is known to be unique) with its local file names. This is called location-dependent naming. It is appropriate for files whose identity is naturally tied to a particular location and in which interest is limited to a small area. However, it is inappropriate for files in which there is enterprise-wide, or cross-enterprise, interest. It is especially inappropriate for files that are moved from one location to another or that have copies at multiple locations.

The best global name assignments are location independent. The identity of every file can be tied to some higher-level grouping—if not location, perhaps organization, or country. The technique used by SNA/FS is to allow any higher-level grouping to be used. Location is just one of many higher-level groupings possible in SNA/FS global names.

SNA/FS global names consist of a string of tokens, arranged in hierarchical order, with the leftmost the highest, or root, token. The values for each token position are qualified by the higher-order token on its left and, therefore, need be unique only within that token.

Values for the leftmost tokens are assigned by SNA. Assignment ensures the value is unique and registers the value and the identity of its owner. For example, MCODE has been assigned to SNA/Management Services (SNA/MS).[7] Since SNA/MS owns that value,

when the leftmost token is MCODE, SNA/MS is responsible for assigning the values for the token immediately to the right. For the tokens further to the right, SNA/MS can either assign values or delegate the responsibility to some appropriate authority.

Appropriately designed name strings can make the delegation of responsibility reasonably straightforward. For example, in the name string MCODE.9135.MOD2.MAINTEC.12345 the second token position contains the IBM machine number, and the fifth contains the maintenance EC numbers for the 9135 machine. Both of these numbers are administered by an assigning authority, in this case certain departments within IBM, whose responsibilities include the administration of those numbers. The fact that their numbers are incorporated into SNA/MS object names adds nothing to their already existing responsibilities. Similarly, many other kinds of objects will be found to already have some sort of distinguishing serial number that can be incorporated into the name string.

The notions of structuring a name into a string of hierarchical tokens and using that structured name to achieve location independence are becoming more commonly accepted throughout the industry. It is hoped that standards organizations will take on the responsibility for managing more token spaces with the eventual result that the same object names could be used in a variety of network types.

The catalog

The advantages of globally unique, system-independent names for all files and data objects in a network, or any collection of interconnected networks, are very important. For existing systems, global naming is achieved at the price of an extra level of catalog. A catalog is a listing of the names of files and data objects at a location. The traditional catalog contains local names and supports references to the files by local names. The additional level of catalog contains the mapping of global to local names and supports references to those files by global name. Existing systems will continue to support their own local names. Every source node must convert its local name to the global name before sending a file; every target node must do the reverse before storing it. For new systems with few locally created files, local naming can be completely avoided. All references are by global name, and a single level of catalog containing only global names is all that is needed.

Figure 2 depicts the situation in a network just as file MCODE.9135.MOD2.FIX.12345 leaves the source node. The source node in this example keeps a variety of different files in a large holding area. Its catalog maps the data-object names to a local member name within the holding area. The target node, in contrast, keeps each file as a separate entity. Its catalog contains individual local addresses for each file.

A data object cannot be updated or changed in any way.

The SNA/DS distribution is depicted by the large arrow flowing from the source to the target. It contains, among other things, the global name and the data-object contents. It does not contain any local names, source, or target. Only the global name flows in the distribution. The distribution also contains an instruction for the target node to create and load a "new" file. The file will be new to the target node. From the perspective of the enterprise, however, an existing file is being replicated at the target location.

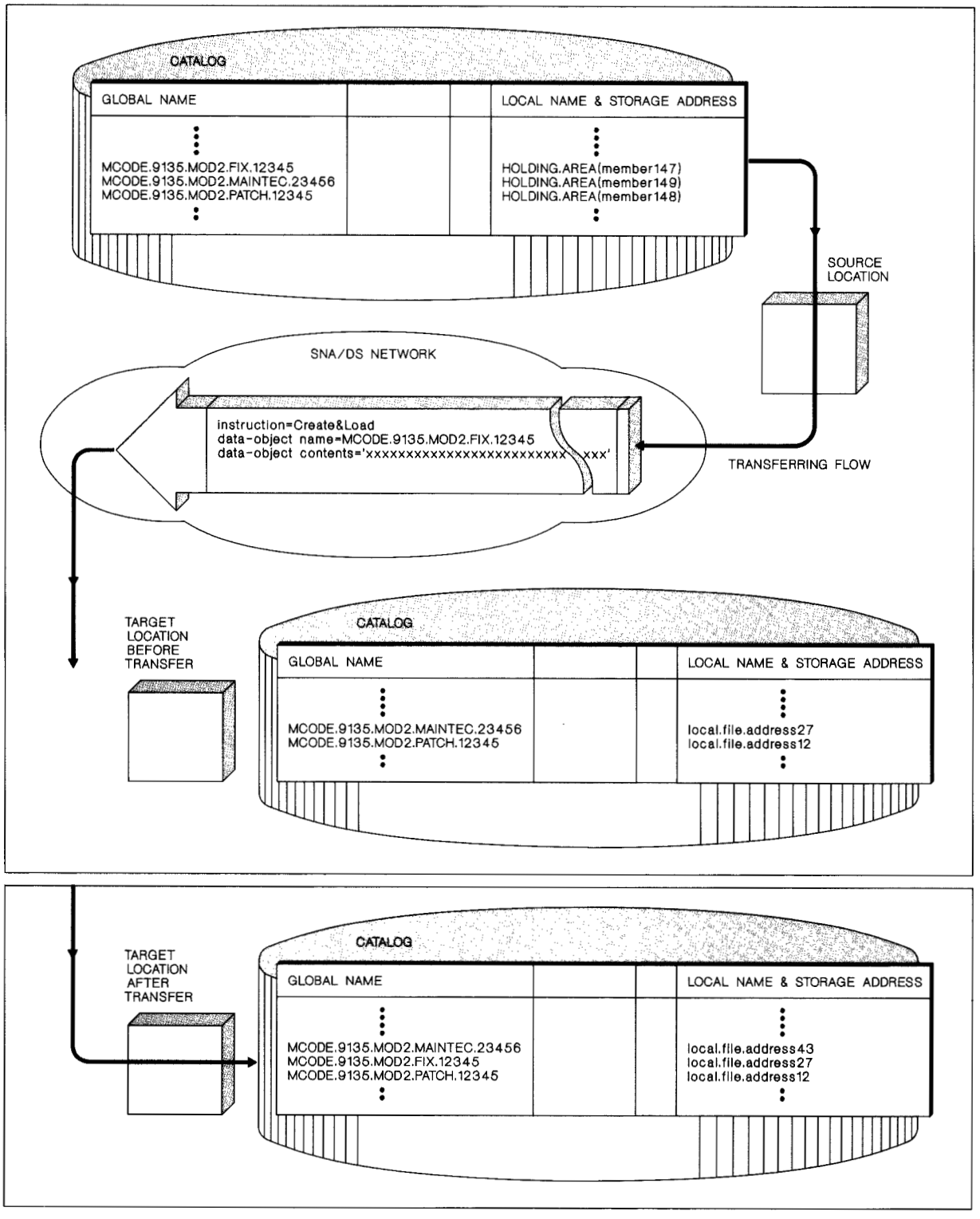
The target catalog, shown before the distribution arrives, does not contain a copy of the file it is about to receive. After the file has been received, a new entry is inserted into the target catalog, as shown after the transfer is complete. The local addresses are assigned by the target system, ensuring no duplication with existing local files. The copy of a file and its catalog entry at the source remain unchanged throughout the process.

Partial name processing

The SNA/FS mechanism for identifying and controlling different versions of updated files is called *partial name processing*. It depends upon the SNA/FS notion of a data object, identified by the complete global name, and the traditional notion of a file, identified by part of the global name.

In SNA/FS, a data object is a named entity that cannot be updated or changed in any way. If the smallest

Figure 2 Example of catalog changes when transferring a file



change is made to the bytestream constituting a data object, it is no longer the same data object and cannot be identified by the same name. In other words, the slightest change to what was one data object creates a new and different data object that requires a new and different name. However, the difference in names may be very small. For example, in most systems when a file is edited and then saved, the date and time at which the file was saved are captured and kept with the file. In SNA/FS, the date and time would be considered to be part of the data-object name, and the freshly stored version of the file would be a different data object with a different name, because the date and time values would not be the same. This precise definition of a data object allows SNA/FS to ensure that all copies of the object identified by a global name are absolutely identical.

Partial naming allows users to express their concept of a file when they design their global names.

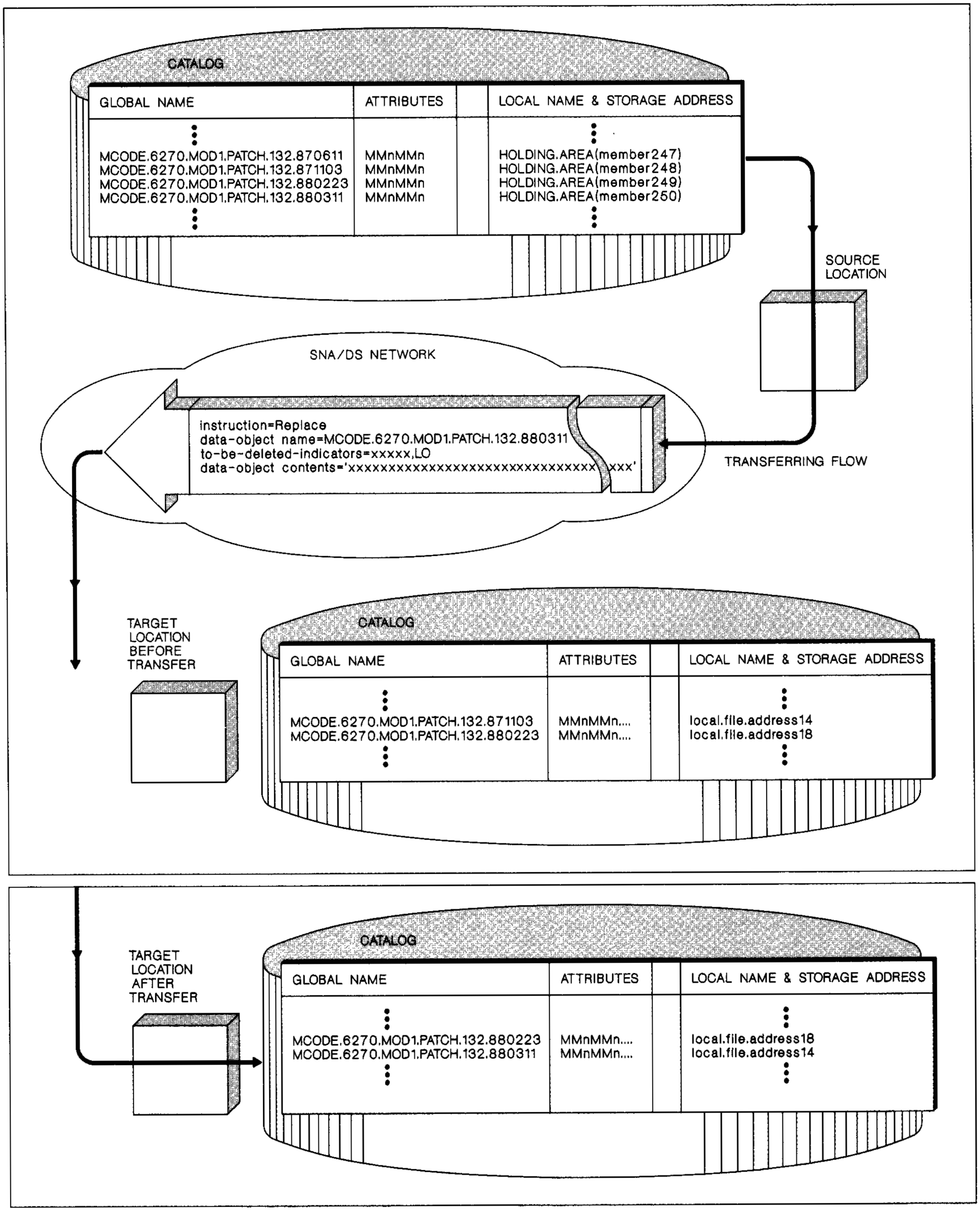
However, users often tend to think of their files as ongoing entities that are essentially the same thing even though they may be frequently updated. In the example of the time-stamped file, the user would wish to use the same name every time he or she edited it. In fact, it would be extremely inconvenient for the user to have to key in the precise date and time of the last update. SNA/FS resolves this conflict between the needs for naming ever-changing files and unchangeable data objects by the notion of partial naming. SNA/FS does not precisely define the concept of a file. Different systems and different applications define their files differently. Partial naming copes with these differences by allowing the part of the global name that identifies whatever the user thinks of as a file to be defined by that user. In other words, partial naming allows users to express their concept of a file in terms of SNA/FS data objects when they design their global names. In practice, one systems administrator would design data-object naming conventions for collections of files for groups of users where both users and files were spread over many locations.

Designing a data-object naming convention requires that each token position be designated as Must Match or Need Not Match. Typically, tokens such as date and time or version number would be designated Need Not Match, and tokens that the user wished to identify the file with would be Must Match. If only one version of a file existed, the Must Match tokens alone would suffice to identify it. If two versions of the same file existed, the Must Match tokens alone would match with both. When SNA/FS discovers that multiple names match on Must Match tokens, it uses additional matching information and the Need Not Match tokens to identify the particular file. In a properly designed naming convention, one or more of the Need Not Match tokens contain a tie-breaking value, such as date or version number, that always increases with time. A tie-breaking value allows the user to request the oldest (or newest) version without having to know its precise Need Not Match token value. Instead of specifying a token value, the user supplies a matching indicator, either Select Low (the oldest) or Select High (the newest) for whichever token position contains the tie-breaking version number.

When a file has been changed and the user wishes to save it, it must be assigned a partially different name because the changed file is a different data object. When a replacement is made, SNA/FS requires that the new data-object name match all of the Must Match tokens in the old name. Therefore, the name must contain at least one Need Not Match token, and the different data object must have a different value for that token. Although users may think that one version of their file is replacing another, in terms of the data objects constituting that file, one data object is deleted and another is created. From an SNA/FS perspective, the two data objects are unrelated except for their common Must Match tokens. Therefore, when replacing is being done, a requester must identify both the new data object and the data object to be deleted. Since the names must be at least partially identical, the requester needs to supply one string of the common, Must Match, tokens plus enough additional information about the objects individually for SNA/FS to make the distinction between them.

In the example illustrated in Figure 3, all of the data-object names have one token (the sixth) that contains

Figure 3 Example of catalog changes when replacing a file



a version date. In both the source and target directories, all of the data-object names within that name collection have token attributes of MMnMMn. This means that the first and second tokens are Must Match, the third is Need Not Match, and so on. The attributes of each name are stored in the catalog along with the name itself. The diagram includes the SNA/DS message unit that would flow from the source to the target. In this case, the SNA/FS server instruction is Replace. The Replace instruction contains the complete name of the new data object plus a set of To Be Deleted indicators that, when combined with the name of the new object, will suffice to identify the old object to be deleted. The target catalog is shown both before the message unit arrives, when 871103 and 880223 are in storage, and after the replacing operation is completed, when only two versions of this file are still at the target, but now they are 880223 and 880311. In other words, the Replace instruction and the information to be deleted have resulted in 880311 replacing 871103.

Need Not Match tokens are not limited to arithmetic values. For example, users could control their own versions with names such as MYPROG.OLDEST, MYPROG.LESSOLD, and MYPROG.NEW, where the token on the right was designated Need Not Match. As another example, global names could include tokens that are meaningful to a human reader when the names are displayed but are not essential to the SNA/FS name-matching process. For example, the first five tokens in the strings MCODE.9135.MOD2-.FIX.12345.PRINTER and MCODE.9135.MOD2.FIX.23456-.ADAPTER serve to uniquely identify the data objects. The sixth token is designated Need Not Match. It is not required to make the name unique. However, it conveys attribute information that is useful to the planner, and having it routinely displayed as part of the object name is convenient.

When creating a data object at a target, a complete global name must be placed in the catalog entry. However, it may be inconvenient for the requester to specify all of the Need Not Match token values precisely. In this case, the requester can specify the name partially and cause the target SNA/FS server to generate and insert the appropriate value for the other tokens. For example, sequential numbers can be automatically incremented, and date and time values can be obtained from the clock of the target.

Classifying files

The tokens used in local names on most systems today often are a mix of instance identifiers, such as serial numbers, and file classification identifiers, such as SCRIPT or TXT. Applications will often define their SNA/FS global names with similar mixtures. For example, the SNA/MS name MCODE.9135.MOD3-.PATCH.12345 contains classification information, such as PATCH, that change-management applications[8] need. However, classifications imbedded in the MCODE global name are only available to applications that understand the MCODE name tree. Applications conforming to a cross-system architecture such as SNA/MS do not find this difficult.

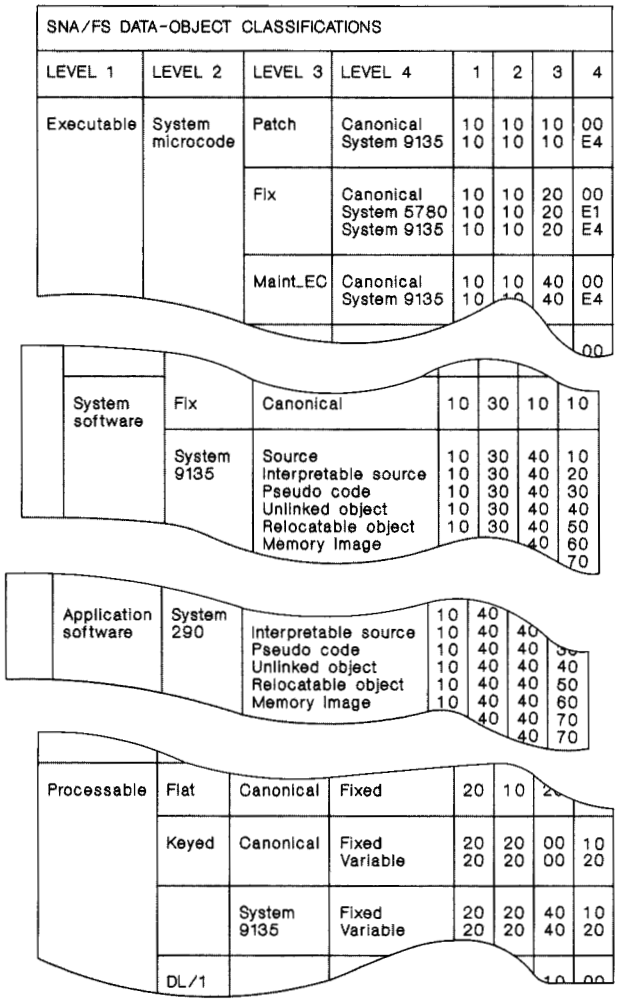
In contrast, programs, such as editors, each of which conforms to its particular operating system, rather than a cross-system architecture, have no global naming convention in common. Furthermore, the classification imbedded in the local name of one system is incompatible with that of other systems. For example, SCRIPT and TXT are values imbedded in two different local name strings. Obviously, the values are different, but more confusingly, some TXT files can be processed by a SCRIPT-capable program, others cannot, and vice versa. If such programs are to exchange the kind of classification information that they need, a common canonical form of classification must be used. SNA/FS defines a classification code for this purpose.

SNA/FS avoids any dependency upon classifications imbedded in data-object names, thereby relieving the global name designers of any requirement to include operating-system-specific name tokens. Users are free to include type application classifications as part of their global name. In fact, it would be difficult to preclude all connotation of classification in a naming convention. However, any such imbedded classifications cannot be recognized by system or SNA/FS facilities.

SNA/FS formally provides file classifications for all files that can appear in FS catalogs, from card images to print fonts to memory dumps. When files are introduced to the SNA/FS world, their classification is supplied to, and subsequently maintained in, the global catalog. Just as the catalog serves to map the global names to local names, it can also support mapping or cross-checking global classification codes to classifications imbedded in the local name, at least for those file classes pertinent to the particular system.

SNA/FS classification codes are particularly useful in identifying classes of files that are common to many systems. Examples of such generic files are process-

Figure 4 Fragments of the data-object classification table



SNA/FS DATA-OBJECT CLASSIFICATIONS							
LEVEL 1	LEVEL 2	LEVEL 3	LEVEL 4	1	2	3	4
Executable	System microcode	Patch	Canonical	10	10	10	00
			System 9135	10	10	10	E4
		Fix	Canonical	10	10	20	00
			System 5780	10	10	20	E1
			System 9135	10	10	20	E4
		Maint_EC	Canonical	10	10	40	00
			System 9135	10	10	40	E4
							00
	System software	Fix	Canonical	10	30	10	10
		System 9135	Source	10	30	40	10
			Interpretable source	10	30	40	20
			Pseudo code	10	30	40	30
			Unlinked object	10	30	40	40
			Relocatable object	10	30	40	50
			Memory Image			40	60
							70
	Application software	System 290	Interpretable source	10	40		
			Pseudo code	10	40	40	
			Unlinked object	10	40	40	40
			Relocatable object	10	40	40	50
			Memory Image	10	40	40	60
					40	40	70
						40	70
Processable	Flat	Canonical	Fixed	20	10	2	
	Keyed	Canonical	Fixed	20	20	00	10
			Variable	20	20	00	20
		System 9135	Fixed	20	20	40	10
			Variable	20	20	40	20
	DL/1					10	00

able flat files and printable documents. One can imagine a file known locally at the source as SCRIPT being converted to the generic SNA/FS classification code, shipped to multiple targets, and being reconverted to TXT at one target and to DOC at another.

Other file classes are specific to a particular system—an executable load module, for example. The hierarchical structure of the classification codes allows for system-, product-, and customer-specific classification as well.

The classification code is permanently associated with a data object. The classification of a data object cannot be changed, just as its global name cannot be changed, without creating a new data object with a new name. However, just as a file is deemed to be the same file even when a new data-object version of it replaces the current one, a file would also be deemed to be the same file even if a new data object of a different name and classification replaces the current one. When copies of the data object are transported to other locations, the classification code accompanies them.

The classification is encoded in a series of registered code points arranged in a hierarchy of four levels. Each level allows 256 codes, so the complete code is mathematically limited to 2^{32} possible values. Registered codepoints save space in message formats and internal tables and facilitate national language support.

The first, or highest-level, byte identifies the major category, which could be executable, processable, presentable, or maintenance information. The meaning of the second, third, and fourth bytes depends upon the value(s) of the higher-level bytes. The table containing all of the assigned code points is large. Figure 4 illustrates some fragments of that table.

Intention. The requester's intention can be useful to the SNA/FS server when it decides whether or not to accept the file. Accordingly, the intention accompanies files to the target as a parameter of the server instruction. The target can determine whether or not to proceed with the receiving and storing operation on the basis of whether the intention is storing, processing, or executing.

Determining file acceptability

The target server can compare the requester's intention, the file classification, and the capabilities of the target system to determine whether or not the file should be accepted. The capabilities of the target system are expressed in a data-object acceptance table. Each system type would have a differently defined table. The fragment of a table shown in Figure 5 illustrates how the capabilities of a fictitious System/290 would be defined for the data-object classes shown in Figure 4.

For example, imagine that a data object classified as 10,30,40,50 was arriving at this fictitious System/290 target location. The acceptance table is scanned from the top down and the "**" entries mean none of the above, so 10,30,40,50 finds a match at 10 ** ** **

If the incoming instruction had an intention of storing, the target would accept the data object because of the "yes" in the storing column of the table. In contrast, the same data-object classification would be rejected if the accompanying intention was executing. Referring back to the classifications in Figure 4, we can see that 10,30,40,50 means System/9135 object code. The only class 10s that the System/290 will accept for executing must begin with 10,40—its own system-specific executable classes.

Use of SNA/Distribution Services

SNA/DS defines the connectionless transport that supports the SNA/FS requirements described above. SNA/DS is a general-purpose communications service that transports application-defined data, known to SNA/DS as "objects," to one or more specified destinations. SNA/DS is completely insensitive to the object contents, which it encapsulates in a *distribution.* A distribution may be stored and forwarded by "intermediate" nodes (i.e., neither the origin nor the destination of the request). A multiple destination distribution will be distributed, that is, copies will be "fanned-out" as it is transported across the network.

At the origin and destination of a distribution, SNA/DS may interact with two application-defined entities—the server and the agent. The distinction between these two entities is made for two reasons:

- SNA/DS interacts with them at different times. The originating agent interacts with SNA/DS at *request* time. Sometime later, at *on-the-fly send time* the server fetches the server object and feeds it piece by piece to SNA/DS as it is sent out over the connection. At the destination, the receiving server stores the server object piece by piece at *on-the-fly receive time.* Sometime later yet, the destination accepts the distribution at *delivery time.*

  This relationship is illustrated in Figure 6. The interaction between the agent and SNA/DS takes place within the node across the SNA/DS request protocol boundary (PB). The interaction between the server and SNA/DS takes place across the SNA/DS server PB.

- One server can provide common function for a variety of agents. Equally, one agent may make use of different servers to assist in the movement of different kinds of objects. Distinguishing between agent and servers permits efficient packaging of function.

Figure 5 Sample data-object acceptance table

DATA-OBJECT ACCEPTANCE TABLE

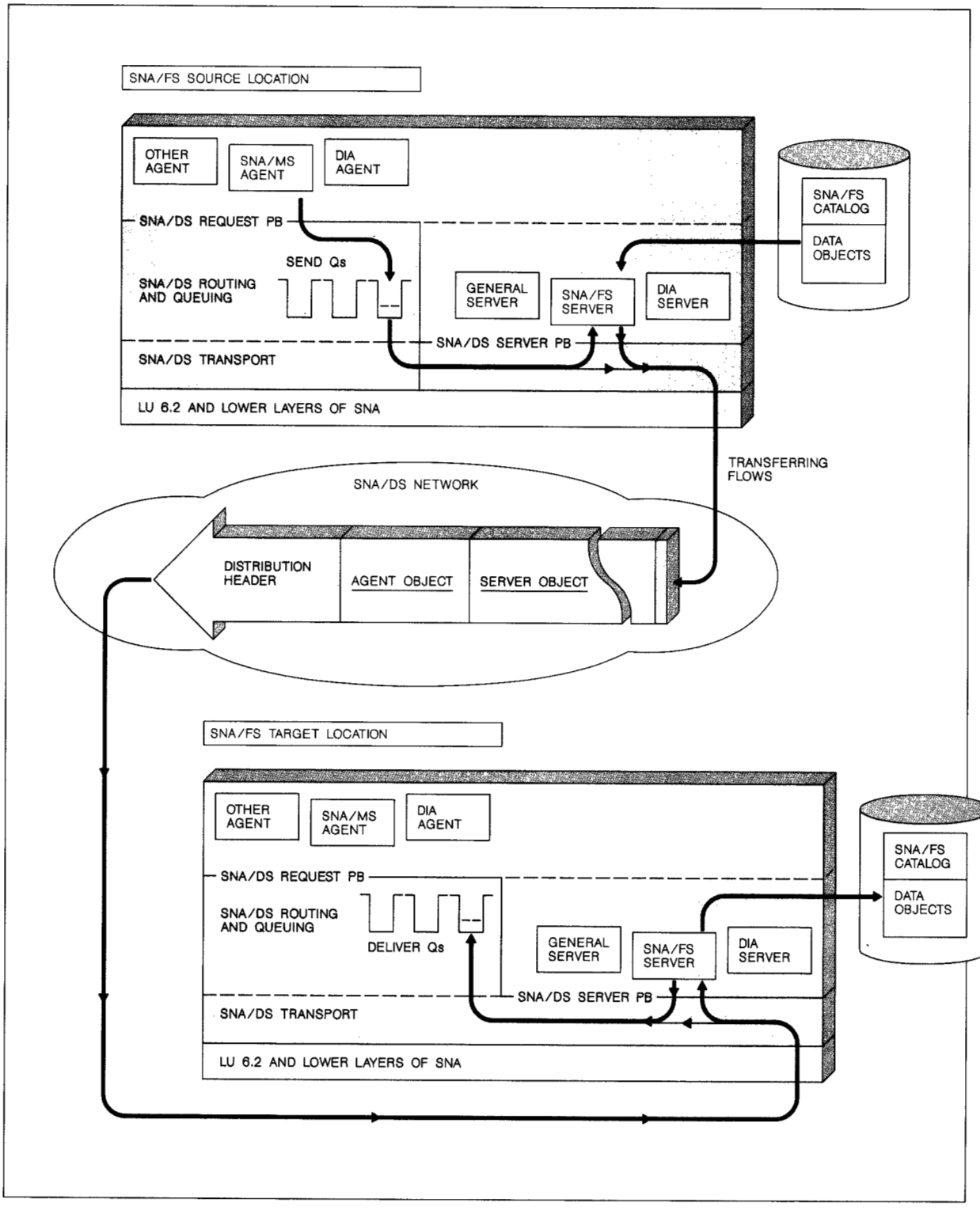
CLASS CODE				INTENTION		
1	2	3	4	EXEC	PROC	STOR
10	40	**	**	YES	NO	YES
10	**	**	**	NO	NO	YES
20	10	**	**	NO	YES	YES
20	20	00	**	NO	YES	YES
20	20	50	**	NO	YES	YES
20	20	**	**	NO	NO	YES
				NO	NO	YES

A distribution can carry two objects, an agent object and a server object. The agent object is limited in size and is accepted directly from the originating agent and delivered directly to the destination agent. The server object is unlimited in size and resides on a nonvolatile storage medium. The requester identifies the server object at *request time,* and it is fetched by the originating server at *on-the-fly time.*

SNA/DS transports the agent object from the originating agent to the destination agent and the server object from the originating server to the destination server. SNA/DS does not transport anything from the originating agent to the destination server or from the originating server to the destination agent. Any interaction between servers and agents takes place within a node.

Figure 6 shows an SNA/MS agent communicating with another SNA/MS agent via the agent object and causing the source SNA/FS server to communicate with the target SNA/FS server. It would be equally possible for other SNA/FS-capable agents to cause the originating SNA/FS server to communicate with the target SNA/FS server. It is the differentiation of agent and server function that allows different applications, which act as agents, to share the common SNA/FS function, which is provided by the one server.

Figure 6 SNA/FS use of SNA/DS



At the origin, the agent includes server instruction information in its request to SNA/DS, which passes it to the server when the time comes to send the distribution. The server needs that information to identify the file or files to be fetched. In certain SNA/FS roles, there are no files to fetch, but some SNA/FS server instruction information is involved in all SNA/FS roles. The agent includes it, expressed in verb operand form, in its request to SNA/DS. At sending time, SNA/DS passes it on to the server, which creates a server object by encoding the information into the SNA/FS-structured format that can be understood by the SNA/FS server at the destination. If files are identified, they are also fetched and encoded into the server object.

At the target, an SNA/FS server receives the server object bytestream and decodes it. If data objects are included, they are stored as specified in the SNA/FS server instructions. The server reports its actions and identifies any files it may have stored by creating a server report, in verb operand form, and passing it to SNA/DS for delivery to the destination agent.

Function and roles of the SNA/FS server

Since each SNA/FS operation involves two or more servers, the requesting agent must be able to explicitly instruct each server, both defining its role and specifying what must be done in that role. At the requester location, the server will simply encode the SNA/FS control information. The instruction for this role is Encode Only. At the source location, the receiving server decodes the SNA/FS control information, as directed by the Decode Only instruction, and then the sending server fetches the data object and encodes it into SNA/FS formats, as directed by the Fetch instruction. Finally, at the target location, the receiving server stores the data object, as directed by one of several possible storing instructions.

Two of the storing instructions, Create&Load and Replace, were illustrated in the examples in Figure 2 and Figure 3. Other server instructions include Delete and Create&Load Or Replace.

SNA/FS-defined server objects. SNA/FS servers generate and understand SNA/FS-defined server objects, which contain the following:

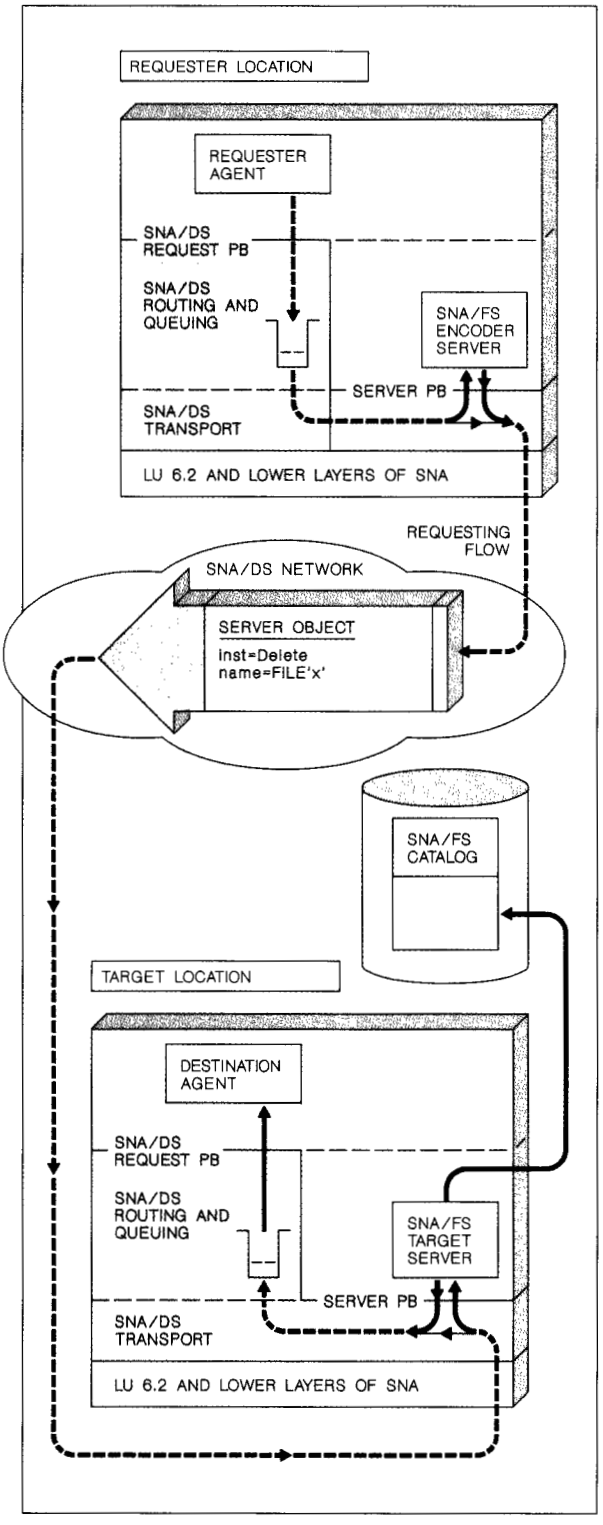
- Instructions—SNA/FS server instructions specify the action requested of the server (e.g., fetch, create, delete). Since an agent request may involve servers at several locations, a server object may contain several server instructions. The first server instruction in the string is performed and then taken off the string. By the time the object arrives at a target location, only one instruction remains.
- Data-object names and attributes—All requests involve one or more global names. In addition, these names are often accompanied by various attribute information, such as data-object classification.
- Data-object contents—These contents occur only in flows between source and target locations. SNA/FS does not define the format of the contents. They may be implementation-specific, such as load modules or memory dumps, or their structure may be specified by an appropriate contents architecture.

A Delete operation. A simple Delete involves only two locations, two server roles, and one distribution flow. In Figure 7 an agent at the requester location specifies data-object "x" is to be deleted at the target location.

Agent roles at source, target, and report-to locations. In the cases in Figure 7, the only agent involvement was at the requester location. However, in each of those cases there was just one flow. The target agent was not expected to take any action, and no agent object was required to convey agent-to-agent communications.

In many cases the requester will need to have function performed by an SNA/FS source or target agent at another location. The requesting agent specifies the services required with an *agent command.* Source locations are commanded to Transfer To Dest List or Transfer To Requester. Target locations are commanded to Report FS Action. For example, if a requester at location A wishes a file to be transferred from location B to locations C and D (see Figure 1), an SNA/FS command Transfer To Dest List flows from A to B. The agent at location B understands the command and accepts responsibility for triggering the requested object transfers. Two SNA/DS distributions are required. In the first distribution, the SNA/DS origin is the SNA/FS requester, and the SNA/DS destination is the SNA/FS source. In the second, the SNA/DS origin is the SNA/FS source, and the SNA/DS destinations are the SNA/FS targets. The example in Figure 1 also illustrates SNA/FS reporting to a fifth location (node E). Two more SNA/DS distributions are needed for the SNA/FS targets to make their reports. A common unit-of-work correlator, assigned by the requesting agent, identifies each of these SNA/DS distributions as belonging to the same SNA/FS-defined task.

Figure 7 A delete operation



Agents that send and receive SNA/FS agent commands must be *SNA/FS-capable* agents. All SNA/FS-capable agents support a common, minimum required set of SNA/FS-defined commands. SNA/FS capability is usually just a minor part of the function of an application agent. Most SNA/FS-capable agents can do more than just these basic SNA/FS functions and are named for the additional function that they do. For example, the SNA/Management Services agent supports several SNA/MS-specific commands such as Install and Remove.

SNA/FS-defined agent objects. SNA/FS-capable agents generate and understand SNA/FS-defined agent objects, each of which contains an SNA/FS command and its parameters. In some cases, it will also contain a summary report. At the SNA/DS origin, the SNA/FS-capable agent encodes the command as part of the agent object. SNA/DS treats the agent object as a bytestream, ignoring its internal structure. At the destination, SNA/DS delivers it to the destination agent in its encoded form. The destination agent decodes the agent object, extracts the command, and performs the specified function.

Not all SNA/FS operations use SNA/FS commands. As shown in Figure 7, SNA/FS operations sometimes have no need for an agent object at all. In those cases the agent might use the agent object entirely for its own purposes.

When the agents need to use the agent object for the purposes of both their own application and SNA/FS, the requesting agent encodes the agent object contents according to its own application definition, which either implies the relevant SNA/FS command or explicitly includes it as a part of the encodings, thereby ensuring that the agent object serves both purposes.

A Retrieve operation. In retrieval operations, the requester and target roles are performed in one location and the source in another. An SNA/FS-capable agent is required at the source location. Four server roles and two distribution flows are needed. The first distribution flow is shown in Figure 8.

Given that the first distribution was successfully received by SNA/DS and its server object was successfully decoded by the SNA/FS server, SNA/DS then delivers it to the source agent. The agent decodes the agent object, extracts the Transfer To Requester command, accepts the decoder server report, and generates the actual transfer request containing the

two remaining server instructions for the source and target as well as the file identifier. These are all placed in the second distribution flow, which is shown in Figure 9.

A Transfer To List operation. This example is a compound of the earlier examples and serves to illustrate all the locations, roles, and flows identified at the beginning of this paper in Figure 1. The SNA/FS-capable agent at the requester location, node A, wants to send copies of file "x" at node B to nodes C and D with completion reports to be sent to node E. The flow is shown in Figure 10.

Given that the process has been correct up to this point, the source agent decodes the agent object, extracts the Transfer To Destination List command, accepts the decoder server report, and generates the actual transfer request, which contains the two remaining server instructions for the source and targets as well as the file identifier. These are all placed in the second distribution flow, shown in Figure 11.

Whether or not the operation is successful, each target agent is responsible for reporting the SNA/FS action to the designated report-to location. The flow from the target 1 location, node C, is shown in Figure 12.

Exception actions. SNA/FS operations will sometimes complete in a manner that is neither clearly successful nor clearly unsuccessful. For example, if 99 out of 100 objects are stored correctly, only the requester can evaluate whether or not this partial success should be preserved. Unfortunately, the requester and the storing operation might be widely separated in time and space. Therefore, in anticipation of possible failure, SNA/FS requires that the requester specify what action the target server should take if an exception occurs.

In the example of 100 objects to be stored, the requester might judge that even if some failed it would still be desirable to have the remainder stored because subsequent recovery procedures would be simpler. If so, the exception action specified would be Continue.

In contrast, the requester might know that unless all 100 objects were successfully stored, the recovery procedures might involve all of the objects. If so, the exception action would be Backout.

Figure 8 A retrieve operation: requester to source flow

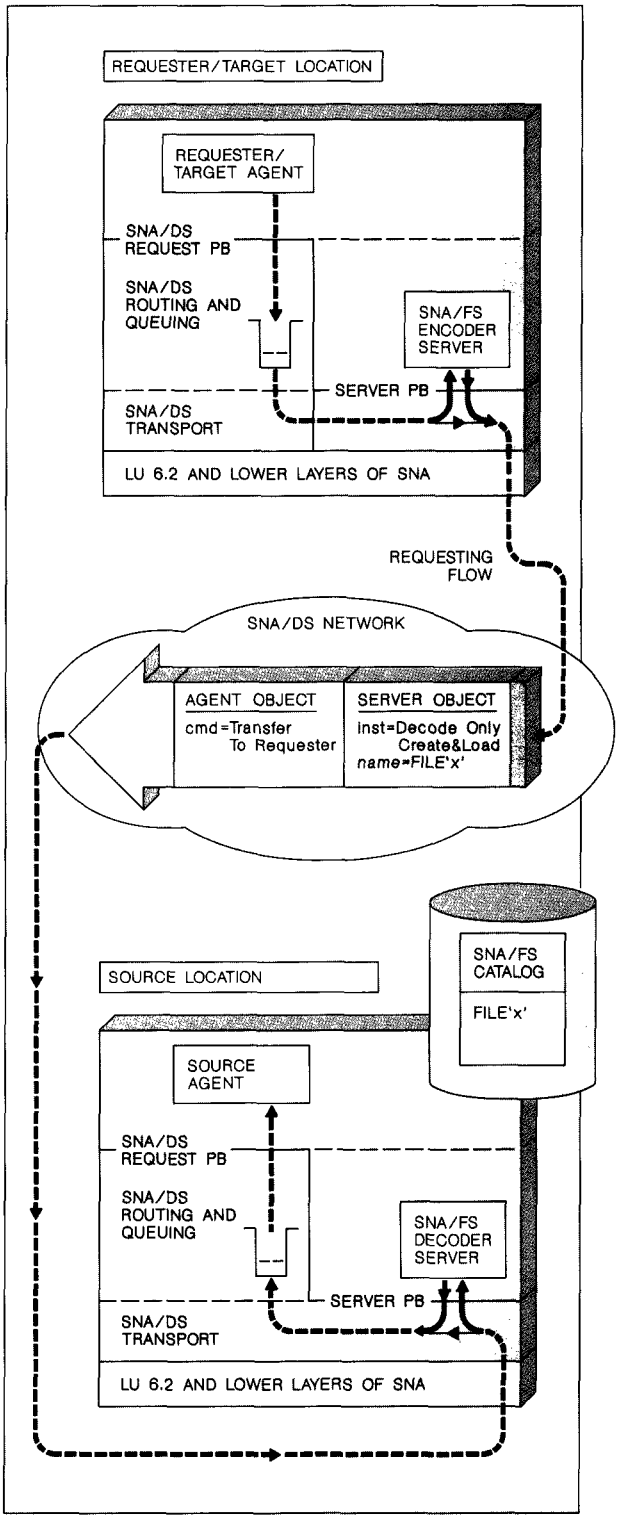
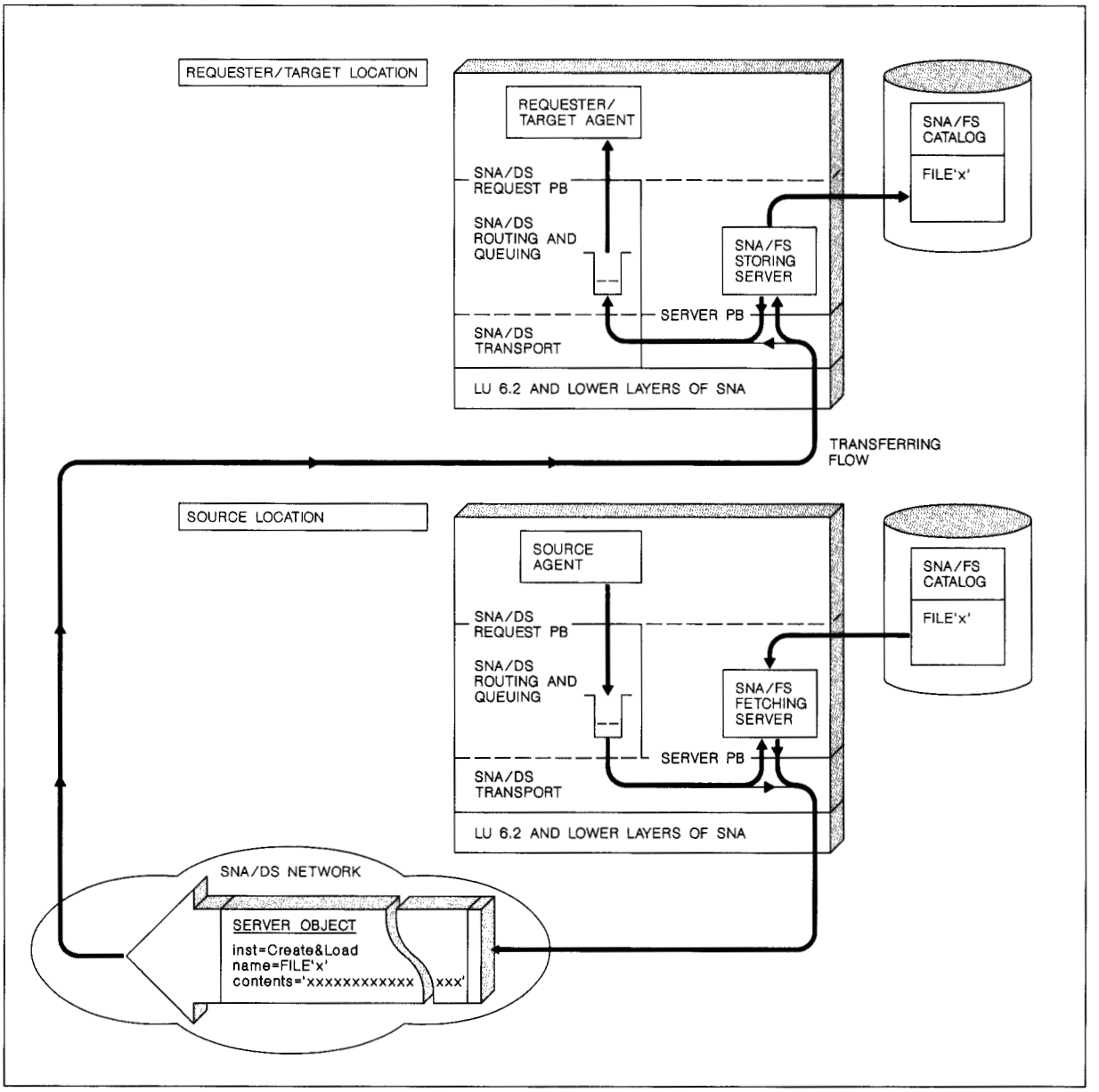


Figure 9 A retrieve operation: source to requester/target flow



Reporting. First of all, the requester specifies whether or not reports are to flow. If they are to flow, the requester specifies where. The report-to agent can be different from that making the request, and the report-to location can be other than the one where the requester resides. If anything occurs that merits reporting, at either the SNA/FS or SNA/DS level, the reports are delivered to the report-to agent at the report-to location.

Only agents can send distributions. When an agent determines that a report is needed, as for example, when a target agent honors the Report FS Action command, it sends a distribution with an agent

object identified by the Reporting FS Action command. Depending on the amount of detail being reported, the distribution may also contain a server object. A server cannot send distributions. An SNA/FS server delivers reporting information to its local agent. The agent includes the server-supplied information in its Reporting FS Action distribution.

The requester can control what level of detail is to be reported by specifying Detailed,Summary Or Exceptions, or Only If Exceptions.

Concluding remarks

This paper has presented SNA/File Services, a component of Systems Network Architecture. SNA/FS and other components of the SNA application layer, including SNA/Management Services and SNA/Distribution Services, work together to provide a variety of application layer function.

SNA/FS defines a global, canonical approach to identifying and moving files in enterprise-wide and industry-wide environments in place of the local, system-specific, techniques historically used by standalone systems. Most SNA/FS function is performed by a file server that can be invoked by SNA/DS or any other application layer component. This arrangement facilitates sharing of the SNA/FS-defined files, data objects, and function by a wide variety of applications.

Acknowledgments

The authors gratefully acknowledge the help of the many designers, developers, and architects involved in this project. Special thanks go to Christopher P. Ballard, David H. Clark, Livio Farfara, Armando Ferrauto, Barbara J. Heldke, Florian K. Kandefer, George M. McMullen, Joseph K. Parks, and Frank J. Petsche for significant design and development contributions, and to Pasquale Autru, James P. Frey, and Matthew L. Hess for welcome management guidance and support.

Cited references

1. *SNA/File Services Reference*, SC31-6807, IBM Corporation (1989); available through IBM branch offices.
2. R. A. Demers, "Distributed files for SAA," *IBM Systems Journal* **27**, No. 3, 348–361 (1988).
3. B. C. Housel and C. J. Scopinich, "SNA Distribution Services," *IBM Systems Journal* **22**, No. 4, 319–343 (1983).
4. *SNA/Distribution Services Reference*, SC30-3098, IBM Corporation (1989); available through IBM branch offices.

Figure 10 Transfer-to-list operation: requester to source flow

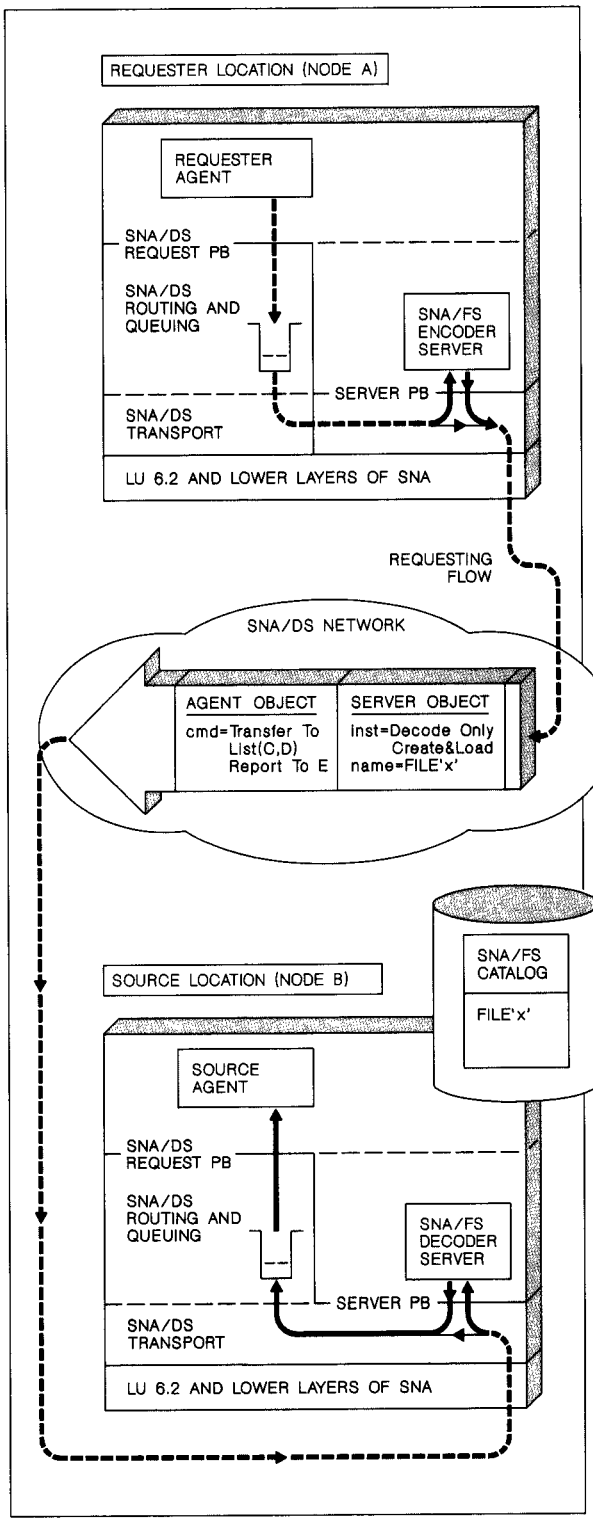


Figure 11 A transfer-to-list operation: source to target flows

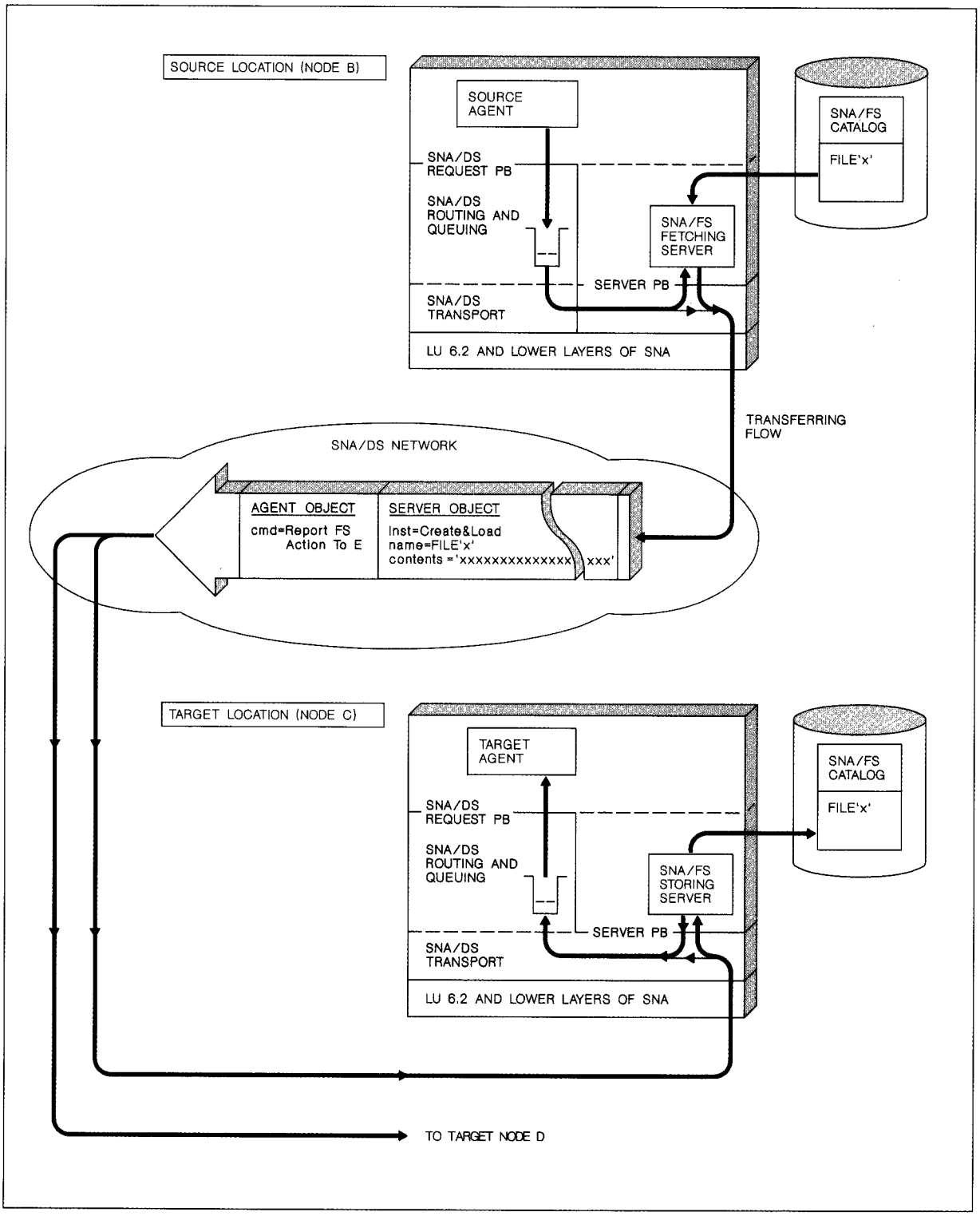
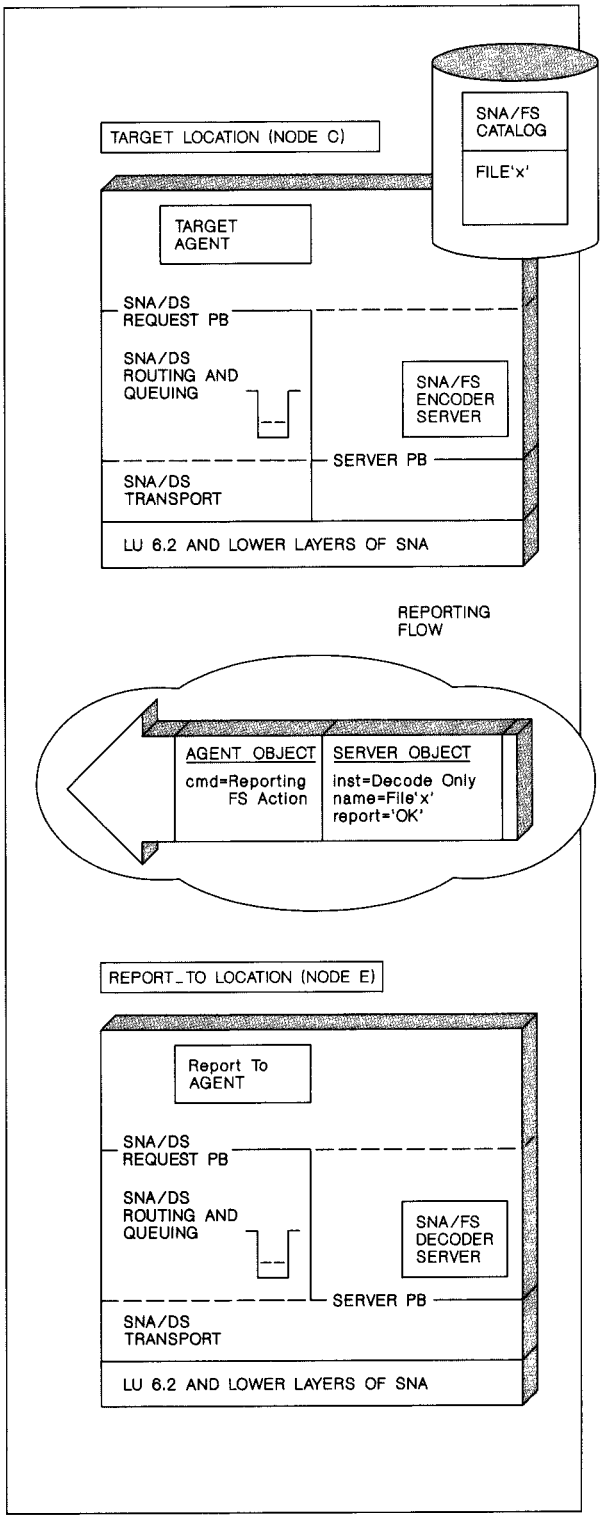


Figure 12 A transfer-to-list operation: target reporting flow



5. R. J. Sundstrom and G. D. Schultz, "SNA's first six years: 1974–1980," *Fifth International Conference on Computer Communication*, Atlanta, GA, North-Holland Publishing Co., Amsterdam (September 1980), pp. 578–585.
6. R. J. Sundstrom, J. B. Staton III, G. D. Schultz, M. L. Hess, G. A Deaton, Jr., L. J. Cole, and R. M. Amy, "SNA: Current requirements and direction," *IBM Systems Journal* **26,** No. 1, 13–36 (1987).
7. *SNA/Management Services Reference*, SC30-3346, IBM Corporation (1989); available through IBM branch offices.
8. C. P. Ballard, L. Farfara, and B. J. Heldke, "Managing changes in SNA networks," *IBM Systems Journal* **28,** No. 2, 260–273 (1989, this issue).

James C. Ashfield *IBM Communication Systems, P.O. Box 12195, Research Triangle Park, North Carolina 27709.* Mr. Ashfield is a senior scientist/engineer. He joined IBM in 1961 and held a variety of positions in marketing for IBM Canada, Americas/Far East, and World Trade. In 1981 he transferred to SNA development. Mr. Ashfield holds a B.Eng. (electrical) from McGill University and an M.B.A. from the University of Western Ontario.

Donna B. Cybrynski *IBM Communication Systems, P.O. Box 12195, Research Triangle Park, North Carolina 27709.* Ms. Cybrynski is currently a staff programmer managing a department responsible for VTAM testing. Ms. Cybrynski joined IBM in 1982 in Management Information Services as a systems analyst. In 1983 she transferred to SNA development, working first on SNA/Distribution Services, then assuming responsibility for the SNA/FS project. She holds a B.Sc. in computer science from the University of Vermont, and an M.Sc. in electrical and computer engineering from North Carolina State University.

Reprint Order No. G321-5357.