

11

THE PRESENTATION AND SESSION LAYERS

The presentation and session layers collaborate to provide many of the distributed-processing capabilities presented to user elements by the service elements of the application layer; for this reason, they are discussed together.

Presentation Layer

Chapter 4 describes how ASN.1 provides the application programmer with a tool for creating data structures that are syntactically independent from the way in which data are stored in a computer and from the way in which they are transferred between computer systems. Transforming these abstract syntaxes into “concrete” data structures appropriate for a given operating system (e.g., UNIX, DOS, VMS, MVS) is typically handled by tools such as *ASN.1 compilers*. The task of preserving the semantics of the data exchanged between a sender and receiver across an OSI network is handled by the presentation layer, which performs the transformations from the local (concrete) syntax used by each application entity to a common *transfer* syntax. This leads us to the discussion of the notion of a presentation *context set*.

Context Set Definition

The presentation layer is responsible for managing the transfer syntaxes associated with the set of abstract syntaxes that will be used by application entities as they exchange information across a presentation connection. As part of presentation connection establishment, application entities must be sure that the presentation layer can support a transfer syntax

for every abstract syntax required by the distributed-processing application(s) that will use this connection. A *presentation context definition list*, consisting of a presentation context identifier and an abstract syntax name, is created by the application entity that initiates the presentation connection establishment. The responding application entity determines whether the list is complete and whether it can be supported, on a “per-entry” basis. The set of presentation contexts resulting from this negotiation is called the *defined context set* (DCS).¹

Presentation Service

The presentation service (ISO/IEC 8882: 1988) is presented in terms of the *facilities* it provides. The *connection establishment* (P-CONNECT) and *connection termination facilities* (P-RELEASE, P-U-ABORT, P-P-ABORT) provide presentation connection management between communicating application entities. Within the context established for a presentation connection, and through the use of the *context management facility* (P-ALTER-CONTEXT), the presentation layer preserves the semantics of data as they are transferred between applications.

In OSI, certain functions reflected by application service elements are performed in the session layer—e.g., token management, synchronization, and checkpointing. When these session layer services are invoked, and because the rigid layering prescribed by the OSI reference model does not allow one to “skip layers,” the presentation layer is seemingly “in the way.” The presentation layer does not provide the service directly but instead passes these service primitives between the application and session layers; thus, the so-called *pass-throughs* were born. For applications that require direct manipulation of session services, the presentation layer offers applications “pass-through” facilities to services offered by the session layer. Collectively identified under the rubric *dialogue control*, they consist of 21 services that directly reflect session layer token management, activity management, data synchronization, and exception reporting services.

(The complete set of primitives and parameters of the presentation service is illustrated in Table 11.1.)

Note that although the presentation layer service primitives suggest that the presentation layer has some active role in providing activity and token management, synchronization, and exception services, this is not

1. A default context is always known by the presentation service provider and the service users. The need for a default context arises (once again) from the need to support the X.410-1984 mode of the OSI Message Handling System; when no presentation protocol is exchanged, the default context is assumed. The *default context name* may also be passed as a parameter during presentation connection establishment.

TABLE 11.1 Presentation Service Primitives

<i>Presentation Primitive</i>	<i>Parameter Name</i>	<i>Request</i>	<i>Indication</i>	<i>Response</i>	<i>Confirm</i>	
P-CONNECT	Calling presentation address	M	M	M		
	Called presentation address	M	M			
	Responding P-address				M	
	P-context definition list	U	C(=)	C		
	P-context definition result list	U	C(=)	C	C(=)	
	Default context name					C(=)
	Default context result					Pass through
	Quality of service	Pass through	Pass through	Pass through	Pass through	Pass through
	Presentation requirements	U	C	C	U	C(=)
	Mode	M	M(=)			
	Session requirements	Pass through	Pass through	Pass through	Pass through	Pass through
	Initial serial number	Pass through	Pass through	Pass through	Pass through	Pass through
	Initial token assignment	Pass through	Pass through	Pass through	Pass through	Pass through
	Session connection identifier	Pass through	Pass through	Pass through	Pass through	Pass through
P-RELEASE	User data	U	C(=)	U	C(=)	
	Result				M(=)	
	User data	U	C(=)	U	C(=)	
	Result				Pass through	
	User data	U	C(=)			
	Provider reason		M			
P-ALTER-CONTEXT	P-context addition list	U	C(=)			
	P-context deletion list	U	C(=)			
	P-context addition result list				U	
	P-context deletion result list				U	
P-DATA	User data	U	C(=)	U	C(=)	
	User data				U	
	User data				U	
P-TYPED-DATA	User data	M	M(=)			
	User data	M	M(=)			
	User data	M	M(=)			
P-EXPEDITED-DATA	User data					
	User data					
	User data					

TABLE 11.1 Presentation Service Primitives continued					
<i>Presentation Primitive</i>	<i>Parameter Name</i>	<i>Request</i>	<i>Indication</i>	<i>Response</i>	<i>Confirm</i>
P-CAPABILITY-DATA	User data	U	C(=)	U	C(=)
T-TOKEN-GIVE	Tokens	Pass through	Pass through		
P-TOKEN-PLEASE	Tokens	Pass through	Pass through		
P-CONTROL-GIVE	User data	U	C(=)		
P-SYNC-MINOR	Type	Pass through	Pass through		
	Synch point serial number	Pass through	Pass through	Pass through	Pass through
	User data	U	C(=)	U	C(=)
P-SYNC-MAJOR	Synch point serial number	Pass through	Pass through		
	User data	U	C(=)	U	C(=)
P-RESYNCHRONIZE	Resynchronize type	Pass through	Pass through		
	Synch point serial number	Pass through	Pass through		
	Tokens	Pass through	Pass through	Pass through	Pass through
	User data	U	C(=)	U	C(=)
P-ACTIVITY-START	P-context identification list		C	C	
	Activity identifier	Pass through	Pass through		
	User data	U	C(=)		
P-ACTIVITY-RESUME	Activity identifier	Pass through	Pass through		
	Old activity identifier	Pass through	Pass through		
	Synch point serial number	Pass through	Pass through		
	Old session connection ID	Pass through	Pass through		
	User data	U	C(=)		
P-ACTIVITY-END	Synch point serial number	Pass through	Pass through	U	C(=)
	User data	U	C(=)		
P-ACTIVITY-INTERRUPT	Reason	Pass through	Pass through		
P-ACTIVITY-DISCARD	Reason	Pass through	Pass through		
P-U-EXCEPTION-REPORT	Reason	Pass through	Pass through		
	User data	U	C(=)		
P-P-EXCEPTION-REPORT	Reason				Pass through

the case. In all instances where the words “Pass through” appear in Table 11.1, all the presentation service provides is access to the corresponding session service. Note also that the presentation service provides (access to) four forms of *information transfer* services: normal, typed, capability, and expedited data. These, too, are best explained in the context of the underlying session service.

Connection Establishment The presentation connection establishment service is invoked via the association control service element (Chapter 10), which supplies presentation (called and calling) addressing information, presentation and session requirements, mode of operation (normal or X.410-1984), and lots more.²

The semantics and composition of the *presentation addresses* used in presentation connection establishment are discussed in Chapter 5; they identify the application entities that will use the presentation connection. Note that if an application entity is represented as a Directory object (see Chapter 7), the ASN.1 definition of a presentation address in an entry of “application entity object” class looks like this:

```

applicationEntity OBJECT-CLASS
SUBCLASS OF top
MUST CONTAIN{
    commonName,
    presentationAddress }
MAY CONTAIN {
    description,
    localityName,
    organizationName,
    organizationUnitName,
    seeAlso,
    supportedApplicationContext }
::= {objectClass 12}

presentationAddress ATTRIBUTE
WITH ATTRIBUTE-SYNTAX
    PresentationAddress
MATCHES FOR EQUALITY
SINGLE VALUE
::= { attribute Type 29 }

```

2. A comparison of Tables 10.1 and 11.1 demonstrates how much of the information required to establish an association between OSI applications “trickles down, percolates up” through several service boundaries. The volume of information that is passed is daunting, and the repetition and cross-referencing between the service and protocol definitions of several layers is one aspect of OSI standardization that contributes to much of the confusion and dismay that afflict the OSI upper layers. The authors hope that readers will be sufficiently comfortable by now with the concepts of the functions provided by the OSI upper layers that the repetition will be recognized as an artifact of the rigid layering approach prescribed by the OSI reference model.

```

PresentationAddress ::= SEQUENCE {
    pSelector      [0]    OCTET STRING OPTIONAL,
    sSelector      [1]    OCTET STRING OPTIONAL,
    tSelector      [2]    OCTET STRING OPTIONAL,
    nAddresses     [3]    SET SIZE (1..MAX) OF OCTET STRING }

```

(Source: ISO/IEC 9594-6: 1990), "Selected Attribute Types."

Presentation requirements are identified in terms of *functional units*, a logical grouping of services somehow distinct from facilities. There are three presentation functional units:

- The *kernel* consists of connection establishment, release, abort, and normal information transfer (P-DATA service only).
- The *context management* facility provides the ability to alter the defined context set (to add or delete a presentation context).
- The *context restoration* facility, used in conjunction with the resynchronization service, allows the defined context set to be recorded at specified points during data transfer. If the presentation connection is resurrected following a temporary failure, the DCS can be restored to one known at the specified "restart" point (i.e., to one that both presentation users remember).

Session requirements, also identified in terms of functional units, represent a checklist of the session services that will be used by the distributed applications. They are provided in the P-CONNECT primitives (see Table 11.1) so that they can be used in the negotiation of a session connection to support the distributed application.



The grouping of presentation services into both facilities and functional units is an artifact of the CCITT/ISO "mind-meld." CCITT speaks of services in terms of facilities, and initially, the presentation service was described in these terms. Later, when CCITT and ISO attempted to merge several notions of the session layer into a single "service," the term functional unit was introduced to better describe the relationship between seemingly disjoint services. Although the presentation service is initially described in terms of facilities, the term is abandoned midway through ISO/IEC 8822 and never used again.

In the *normal mode* of operation, these parameters are conveyed in the connect presentation packet and the negotiated parameter values are returned in the connect presentation accept packet (in Figure 11.1, the CP and CPA PPDUs, respectively). Recall from Chapter 10, however, that no

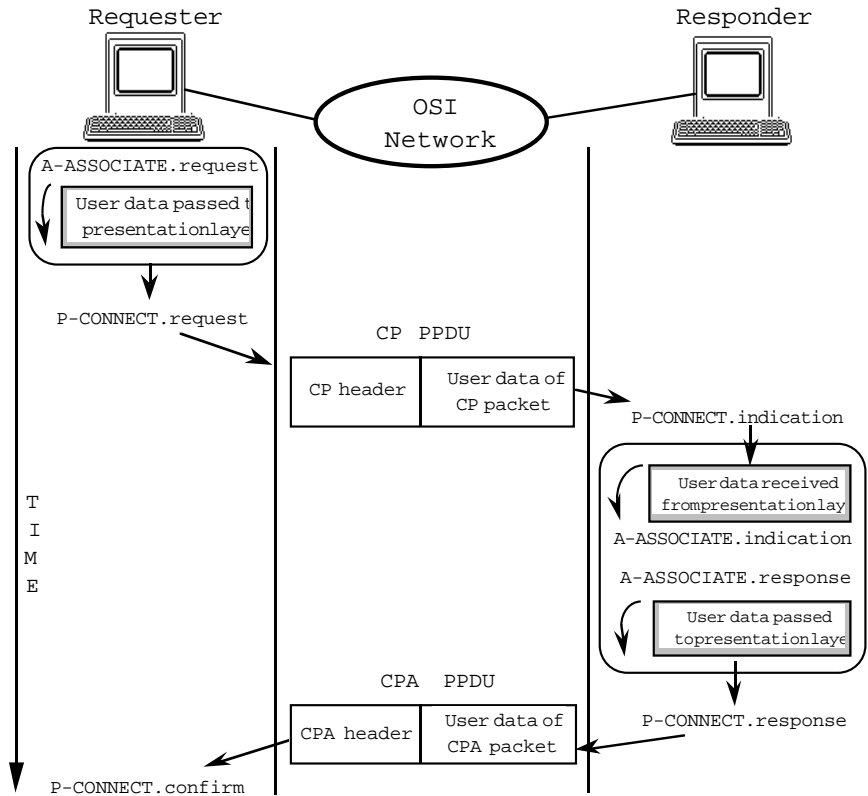


FIGURE 11.1 Presentation Connection Establishment

protocol is exchanged above the session layer to establish an application association with the X.410-1984 mode of operation; under these conditions the presentation context definition list, default-context name, and presentation requirements are always absent from presentation connection establishment primitives, since their values are fixed and understood *a priori* by implementations that support the 1984 version of X.400 and are not needed to create a CP or CPA PDU. The connect presentation reject packet (the CPR PDU, not shown) is used by the presentation service provider or the called application entity to refuse the presentation connection. (If the provider refuses the connection request, it offers a reason—default context not supported, incorrect protocol version, etc.—in the CPR PDU.) The application protocol associated with association establishment (the AARQ and AARE APDUs; see Chapter 10) are typically submitted as user data—a presentation data value in the P-CONNECT primitives,—and “piggybacked” in the presentation protocol, as

illustrated in Figure 11.1.

Many of the parameters used by association control during presentation connection establishment are really intended for use in session connection establishment. Among these are session requirements, session connection identifier, initial assignment of tokens, and initial synch-point serial number. These are used to negotiate a “proper” session connection—one that ensures that all the tools required for communication are present. For example, if the reliable transfer service element is used by an application process, synchronization services and activity management would be included in the session requirements.

Release and Abort Services The presentation layer provides application entities with access to the orderly (nondestructive) release service offered by the session layer; parameters passed in the P-RELEASE service primitives (user data and result, see Table 11.1) are used in the corresponding session release service primitive. In the normal mode of operation, the application protocol used to support the A-RELEASE service is passed directly as user data to the S-RELEASE service; no explicit presentation protocol is used. (In Figure 11.2, the RLRQ and RLRE APDUs are conveyed as user data in the session finish packet, the FN SPDU.)

The abort service operates in much the same manner as the presentation release service. The application protocol used to support the A-ABORT service is conveyed as user data in P-U-ABORT primitives and subsequently transferred in the abort release user packet (in Figure 11.3, the ARU PPDU). The presentation provider may also abort the presentation connection if a protocol error is detected; here, the abort release provider packet (the ARP PPDU, not shown) is sent by the presentation entity that detected the error. (This, of course, begs the question of whether the presentation entity that generated an erroneous packet remains capable of interpreting the ARP packet, but it does provide closure.)

Context Set Negotiation When a presentation connection is established for a single application service element—the message transfer service element of the OSI Message-Handling System, for example—the negotiation process is a formality: all entries in the presentation-context definition list should be supported. When an attempt is made to allow two or more application service elements to share a single presentation connection, the situation becomes more complex. If, for example, the initiating user element wants to allow both the Directory and the common management information service to operate over a single presentation connection, but the responder doesn’t support the directory service, the responder can

TABLE 11.2 Session Service Primitives

<i>Session Primitive</i>	<i>Parameter Name</i>	<i>Request</i>	<i>Indication</i>	<i>Response</i>	<i>Confirm</i>
S-CONNECT	Session connection identifier	U	C(=)	U	C(=)
	Calling session address	M	M		
	Called session address	M	M		
	Responding session address			M	M
	Result			M	M
	Quality of service	M	M	M	M
	Session requirements	M	M(=)	M	M(=)
	Initial serial number	C	C(=)	C	C(=)
	Initial token assignment	C	C(=)	C	C(=)
	User data	U	C(=)	U	U
	User data	U	C(=)	U	U
	Result			M	M(=)
	User data	U	C(=)		
	Provider reason			M	
S-RELEASE	User data	M	M(=)		
	User data	M	M(=)		
	User data	M	M(=)		
	User data	U	C(=)	U	C(=)
	Tokens	M	M(=)		
	Tokens	M	M(=)		
	User data	U	C(=)		
	Type	M	M(=)		
	Synch point serial number	M	M(=)	M	M(=)
	User data	U	C(=)	U	C(=)
	Synch point serial number	M	M(=)		
	User data	U	C(=)	U	C(=)
	Resynchronize type	M	M(=)		
	Synch point serial number	C	M(=)	M	M(=)
Tokens	C	C(=)	C	C(=)	
User data	U	C(=)	U	C(=)	
S-S-ABORT	Session connection identifier	U	C(=)	U	C(=)
	Calling session address	M	M		
	Called session address	M	M		
	Responding session address			M	M
	Result			M	M
	Quality of service	M	M	M	M
	Session requirements	M	M(=)	M	M(=)
	Initial serial number	C	C(=)	C	C(=)
	Initial token assignment	C	C(=)	C	C(=)
	User data	U	C(=)	U	U
	User data	U	C(=)	U	U
	Result			M	M(=)
	User data	U	C(=)		
	Provider reason			M	
S-SYNC-MAJOR	User data	M	M(=)		
	User data	M	M(=)		
	User data	M	M(=)		
	User data	U	C(=)	U	C(=)
	Tokens	M	M(=)		
	Tokens	M	M(=)		
	User data	U	C(=)		
	Type	M	M(=)		
	Synch point serial number	M	M(=)	M	M(=)
	User data	U	C(=)	U	C(=)
	Synch point serial number	M	M(=)		
	User data	U	C(=)	U	C(=)
	Resynchronize type	M	M(=)		
	Synch point serial number	C	M(=)	M	M(=)
Tokens	C	C(=)	C	C(=)	
User data	U	C(=)	U	C(=)	
S-RESYNCHRONIZE	Session connection identifier	U	C(=)	U	C(=)
	Calling session address	M	M		
	Called session address	M	M		
	Responding session address			M	M
	Result			M	M
	Quality of service	M	M	M	M
	Session requirements	M	M(=)	M	M(=)
	Initial serial number	C	C(=)	C	C(=)
	Initial token assignment	C	C(=)	C	C(=)
	User data	U	C(=)	U	U
	User data	U	C(=)	U	U
	Result			M	M(=)
	User data	U	C(=)		
	Provider reason			M	

TABLE 11.2 Session Service Primitives continued

<i>Session Primitive</i>	<i>Parameter Name</i>	<i>Request</i>	<i>Indication</i>	<i>Response</i>	<i>Confirm</i>
S-U-EXCEPTION-REPORT	S-context identification list		C		
	Reason	M	M(=)	C	
S-S-EXCEPTION-REPORT	User data	U	C(=)		
	Reason				M
S-ACTIVITY-START	Activity identifier	M	M(=)		
	User data	U	C(=)		
S-ACTIVITY-RESUME	Activity identifier	M	M(=)		
	Old activity identifier	M	M(=)		
	Synch point serial number	M	M(=)		
	Old session connection ID	M	M(=)		
	User data	U	C(=)		
	Synch point serial number	M	M(=)		
	User data	U	C(=)		U
S-ACTIVITY-END	Reason	U	C(=)		C(=)
S-ACTIVITY-INTERRUPT	Reason	U	C(=)		
S-ACTIVITY-DISCARD	Reason	U	C(=)		

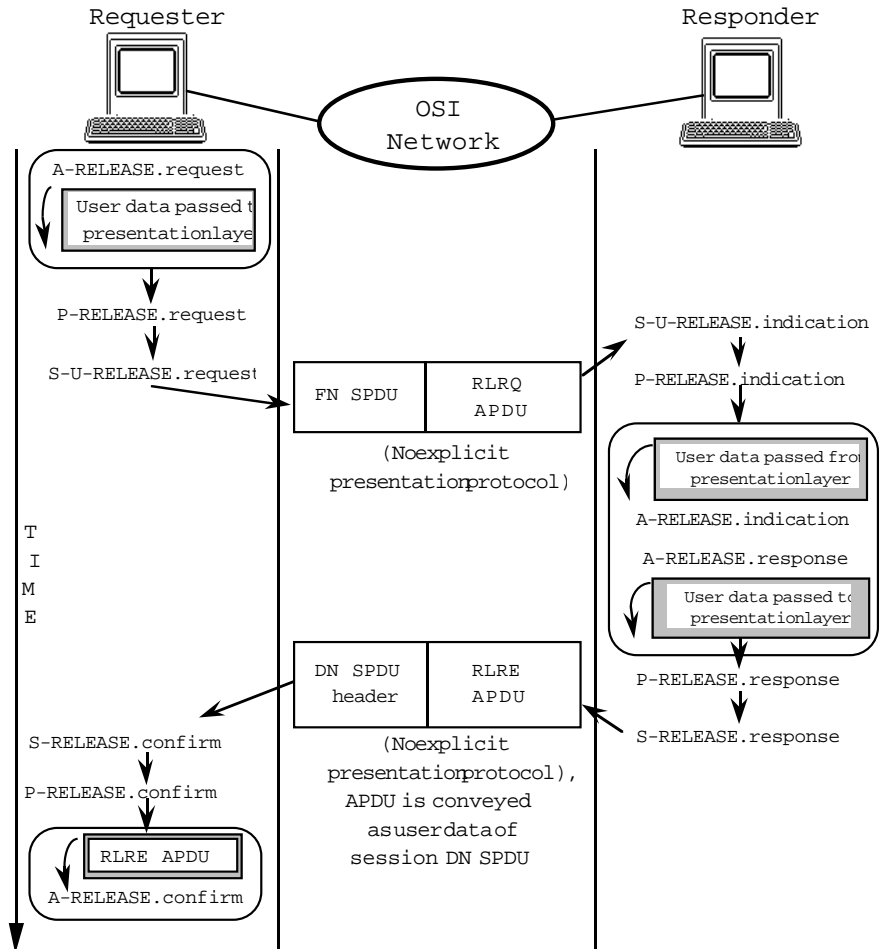


FIGURE 11.2 Normal Release of a Presentation Connection

selectively reject all the entries on the list that are associated with the Directory, and the initiator can determine that only common management information service can be supported over the resulting presentation connection. (Situations such as this are the exception rather than the norm.)

Negotiation and Renegotiation of Transfer Syntax For those situations in which a presentation connection is to be shared or reused by different application service elements, the presentation layer provides an “alter context” service—a user element can add entries to and delete entries from the defined context set (using the presentation context addi-

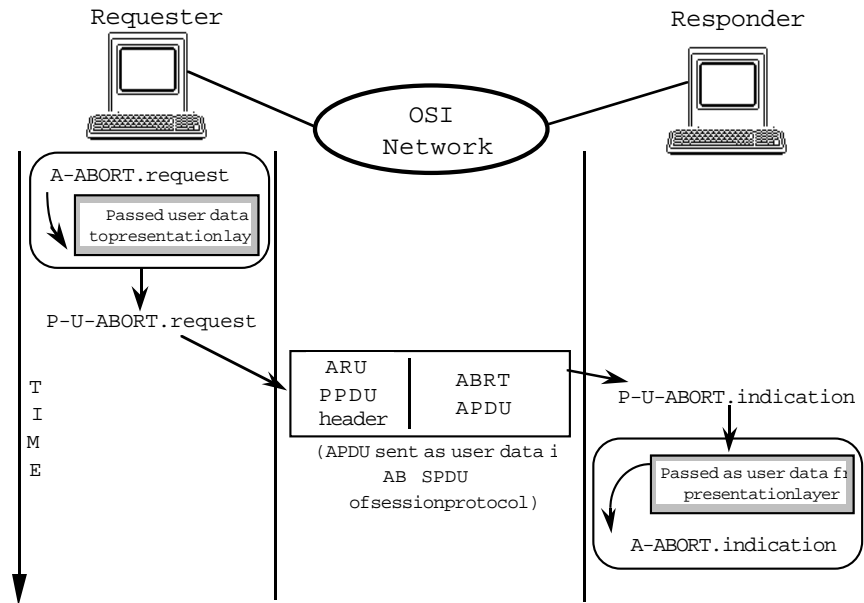


FIGURE 11.3 User-initiated Abort of a Presentation Connection

tion list and presentation context deletion list parameters, respectively). When is this necessary? Consider the scenario in which an FTAM association exists between two open systems, and a user on one system decides to make use of the OSI Directory. Using the P-ALTER-CONTEXT service, the necessary presentation contexts can be added to the defined context set of the presentation connection that already exists. The initiator sends the revised presentation context in an alter context packet, and the responder acknowledges the revisions using the alter context acknowledgment packet. In Figure 11.4, these are the AC and ACA PPDU's, respectively.

Is this useful? It's yet another form of multiplexing, and only time and experience will demonstrate whether the savings in performance is worth the implementation complexity.

Data Transformation to and from Transfer Syntax Transforming data to and from transfer syntax is the basic and essential service of the presentation layer. Data submitted by a sending application service element are transformed from the local syntax to the common *transfer* syntax, transferred from the source system to the destination system, then transformed into the local syntax understood by the receiving application service element at the destination system. This is performed for all transfers

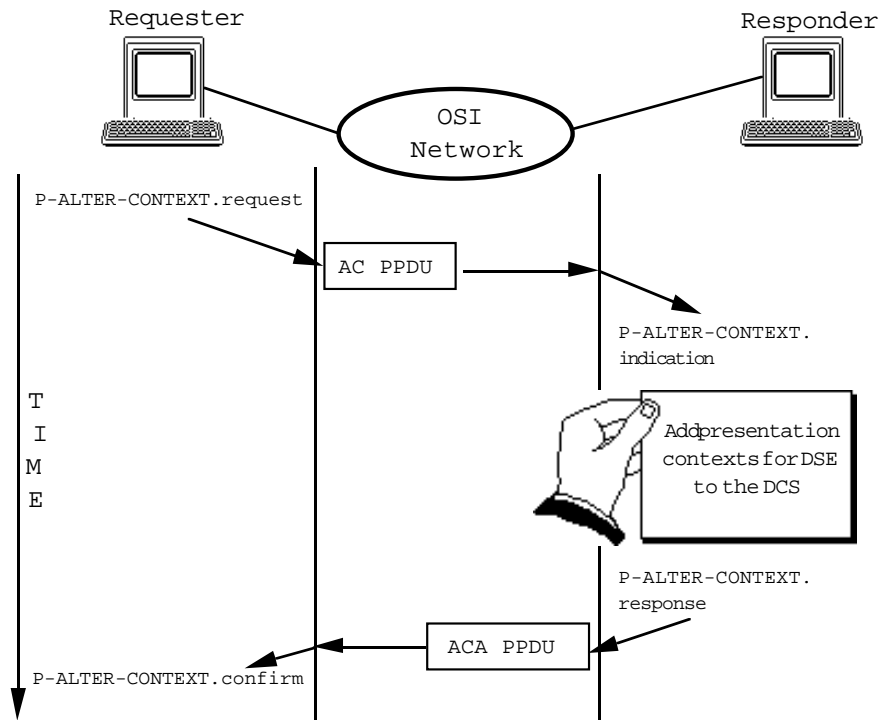


FIGURE 11.4 Changing the Defined Context Set

of presentation data values (user data). This encoding/decoding is normally provided in one of two ways: either the application provides for encoding/decoding itself, perhaps assisted by ASN.1 compiler tools, or the encoding/decoding is provided within the presentation layer. (Although the second alternative would seem to be the obvious choice, the first is more efficient because it avoids “double encoding”: encode data in the local syntax, then in the transfer syntax.)

Information Transfer, Dialogue Control The intention to use session services—including mode of data transfer (duplex, half-duplex), activity management, and synchronization services—is indicated in the `P-CONNECT.request` primitive (see Table 11.1). This is the setup process for an application to use pass-through services. The list of required session services continues on its “trickle down, cross the network, percolate up” journey from the initiating user element to the target user element in two forms. Information is passed to the presentation entity to request “additional” session services during session connection establishment, dis-

cussed later in this chapter. The session service requirements of the requester may also be conveyed in the optional *User session requirements* field (Figure 11.5) of the connect presentation packet (in Figure 11.6, the CP PDU) by setting the appropriate bit of the ASN.1 BITSTRING data structure to a value of 1.

```

User-session-requirements ::= BIT STRING {
    half-duplex           (0),
    duplex               (1),
    expedited-data       (2),
    minor-synchronize    (3),
    major-synchronize    (4),
    resynchronize        (5),
    activity-management   (6),
    negotiated-release    (7),
    capability-data       (8),
    exceptions           (9),
    typed-data           (10) }

```

(Source: ISO/IEC 8823: 1988)

FIGURE 11.5 Abstract Syntax of User-session-requirements field of the Connect Presentation Packet

If session synchronization services are requested, an *initial serial number* may be provided (a session user option; see the discussion of session synchronization services, later in this chapter). The initial direction of information transfer—requester side first, acceptor side first, acceptor side chooses—may also be specified to indicate “who speaks first.” These values are used locally, in the creation of an S-CONNECT.request.

When the connection establishment process is completed, the session services negotiated during that process are available via the pass-through presentation services. The following section examines the session layer and its services in more detail.



This all seems rather unnatural. Why not have application service elements perform the very capabilities they present to user elements? The reasons are largely historical and political. Early in the development of OSI, CCITT and ISO agreed that sharing a single reference model for open systems was a good thing. Having two very large, consensus-driven organizations participate in joint development of a single set of protocols and services, however, was not. Setting aside the issue of cultural differences, coordinating—and particularly, sequencing—the development of standards proved daunting from the outset. Committees were formed in both standards bodies for all seven layers (and then some), and these proceeded in parallel; under such cir-

cumstances, it turned out to be impossible to define standards using a top-down or bottom-up approach. At first, coordination seemed a minor inconvenience; the OSI RM, after all, was there to guide all the standards committees. But the OSI RM itself was revised on several occasions to accommodate “pre-OSI” CCITT recommendations such as the T.62 session protocol for teletex communications and the X.25 packet-level protocol. The result? Functions that arguably should lie in the application layer were assigned to the session layer because the X.400 MHS recommendations were “ready to ship” before the application layer structure was completed (see Chapter 8).

How bad is the resultant upper-layer architecture? Although pass-throughs preserve architectural purity, they are inconvenient: application service elements pass parameters for session in procedure calls to presentation, which copies or passes pointers to the same parameters in procedure calls to session, resulting in what one OSI implementer describes as “silly little no-op pass-through routines, slowing down the whole stack. . . .”³ Exaggerated? Only in the sense that in the overall performance of OSI implementations, this is probably not the killer. An alternative? OSI might have adopted the “tool-kit” philosophy from the application-layer structure earlier in the process, but that’s another case of “hindsight.”

Session Layer

Information exchange between computer systems can be viewed as having two fundamental components. The first is information *transfer*: moving the information from its origin or source to its destination. In OSI (and TCP/IP), this aspect of information transfer is the responsibility of the *transport service*. Now, the transport service moves data between open systems *transparently*; that is, the transport service is only responsible for the transfer of an unstructured or “raw” bit stream of information from one open system to another, and it doesn’t know or care where application data begin and end. Applications, however, do. The second fundamental component—preserving the *structure* of data defined by communicating application processes—belongs to the *session service* (ISO/IEC 8326: 1987). Thus, in addition to the expected connection management services—connection establishment, release, and abort—the session layer provides application processes with the synchronization, checkpointing, and resynchronization mechanisms to organize and

3. From an electronic-mail conversation about the presentation layer with Lisa Phifer.

impose structure on data exchanged. These are collectively referred to as *dialogue-control* facilities.

A further aspect of transport service transparency is that the transport service isn't expected to know or care whether transport service users take turns sending data or do so simultaneously, or even whether consecutive data are submitted by the same application. The responsibility for maintaining *control* of the conversation(s) that take place between communicating application processes also belongs to the session service. The session layer provides application processes with the following mechanisms to control the conversations, or *dialogues*, conducted by communicating application processes:

- Facilities that enable applications to “take turns” exchanging data or otherwise influence information exchange. The coordination of a *half-duplex* method of information exchange, synchronization services, and activity management services are governed by the assignment of *tokens*. Token-based facilities also enable application processes to coordinate the “graceful” release of associations; they enable applications to make sure that all information transfers are complete before their communication is terminated (if this was the way they chose to behave).
- Facilities that allow application processes to conduct several exchanges in the context of a single session connection. Since the unit of application exchange is called an activity (see Chapter 10), the facilities that enable applications to “distinguish between different pieces of logical work” (ISO/IEC 8326: 1987) are collectively referred to as *activity management*.

Session Services

There are more than 20 session services (see Table 11.2). For identification during connection negotiation and also for convenient identification by other standards (notably the presentation service, where session service requirements must be indicated by application entities during association establishment), session services that are related are logically grouped into *functional units*. Each functional unit is examined separately in the following sections.

The Kernel Functional Unit The *kernel* functional unit consists of session connection establishment; “normal” data transfer (half- or full-duplex); graceful, or orderly, release; and user- and provider-initiated abort services. All session connections require use of the kernel. During *connection establishment*, the availability of the session services required by the application entities that will use the session connection is deter-

mined by the session service users (the presentation entities).

There are actually four steps associated with the session functional unit *negotiation* process, and the actions performed are best described by examining the OSI upper layers together:

1. Formally, the calling application entity invokes the association control, in the process identifying the application's session requirements for communication (see Chapter 10); these are "trickled down" to the presentation entity in the P-CONNECT.request and to the session entity in the S-CONNECT.request. In practice, this might be implemented as a series of procedure calls within an "association/connection management" module, with each call passing as a "user data" parameter the protocol data unit(s) of the higher-layer entity. Armed with the cascaded set of CONNECT.request packets, the calling session entity establishes a transport connection (see Chapter 12), and a session connect packet (a CN SPDU) is constructed and submitted to the transport layer in a data request. If shorter than 512 octets, the connect presentation packet (the CP PPDU) is conveyed in the user data parameter of the session connect packet; if between 513 and 10,240 octets, it is conveyed in the extended user data parameter of the session connect packet (available in session version 2 only). Otherwise, only the first 10,240 octets of the CP PPDU are sent in the extended user data parameter of the CN SPDU, followed by one or more connect data overflow packets (CDO SPDUs). (Note that Figure 11.6 and this discussion show only the scenario in which everything fits in a single CN SPDU.)
2. The called session entity receives the CN SPDU, parses it, and generates an S-CONNECT.indication, passing among the parameters the set of session functional units indicated by the called presentation entity (and encoded in the CN SPDU), and session service user data (containing the CP PPDU). The called presentation entity extracts the connect presentation packet (the CP PPDU) and uses the encoded data to generate a P-CONNECT.indication (see Figure 11.1 and the accompanying text). The user session requirements from the calling application entity are identified to the called application entity in the indication ("percolating up" of parameters). The called application entity may make adjustments to user session requirements, based on (a) options left for the called application entity (for example, the calling application entity might have proposed both half- and full-duplex modes of transfer, effectively say-

ing to the caller “You choose”) or (b) known limitations to the underlying session implementation (e.g., if a session functional unit requested by the calling application entity is not supported).

If the resulting list still satisfies the application requirements as understood by the called application entity, it uses association control to accept the association, and the revised session requirements are “trickled down” in the presentation and session CONNECT response primitives, again in the manner described in step 1. The revised requirements are encoded in the session accept connection packet (in Figure 11.6, the AC SPDU), which is returned over the transport connection to the calling session entity. Note that the session functional units that will be used over the session connection are defined as the intersection of what the called and calling session entities exchange in the session user requirements field of the CN and AC SPDUs. Negotiation is thus a “whittling down” rather than a “bartering” process.

4. The calling session entity parses the AC SPDU, determines (and records) the session functional units that will be used across the session connection, then percolates the revised set up as per step (2).

This entire sequence is illustrated in Figure 11.6.



Even though the OSI upper layers are formally organized in a hierarchy, from an examination of the information passed through the presentation layer it is obvious that a purely “clinical” interpretation of the interactions among application, presentation, and session entities would be extremely inefficient. Well-behaved implementations of the OSI upper layers frequently lump association, presentation, and session connection management together as a single process that is aware of, for example, the complement of session functional units available in a particular implementation, the available application service elements, and the requirements they impose on both the presentation layer (what presentation syntaxes) and the session layer (what services). In such implementations, connection management is far from the scavenger hunt the standards depict.

The Many Faces of Data Transfer (Full-Duplex, Half-Duplex, Expedited, and Capability Functional Units) There are two modes of normal data transfer: *full-duplex* (both directions at the same time) or *half-duplex* (either direction, only one direction at a time, with the choice of direction being controlled by the session service users). The half-duplex functional unit uses the give tokens and please tokens services (S-TOKEN-GIVE, S-TOKEN-PLEASE): a *data token* is exchanged between

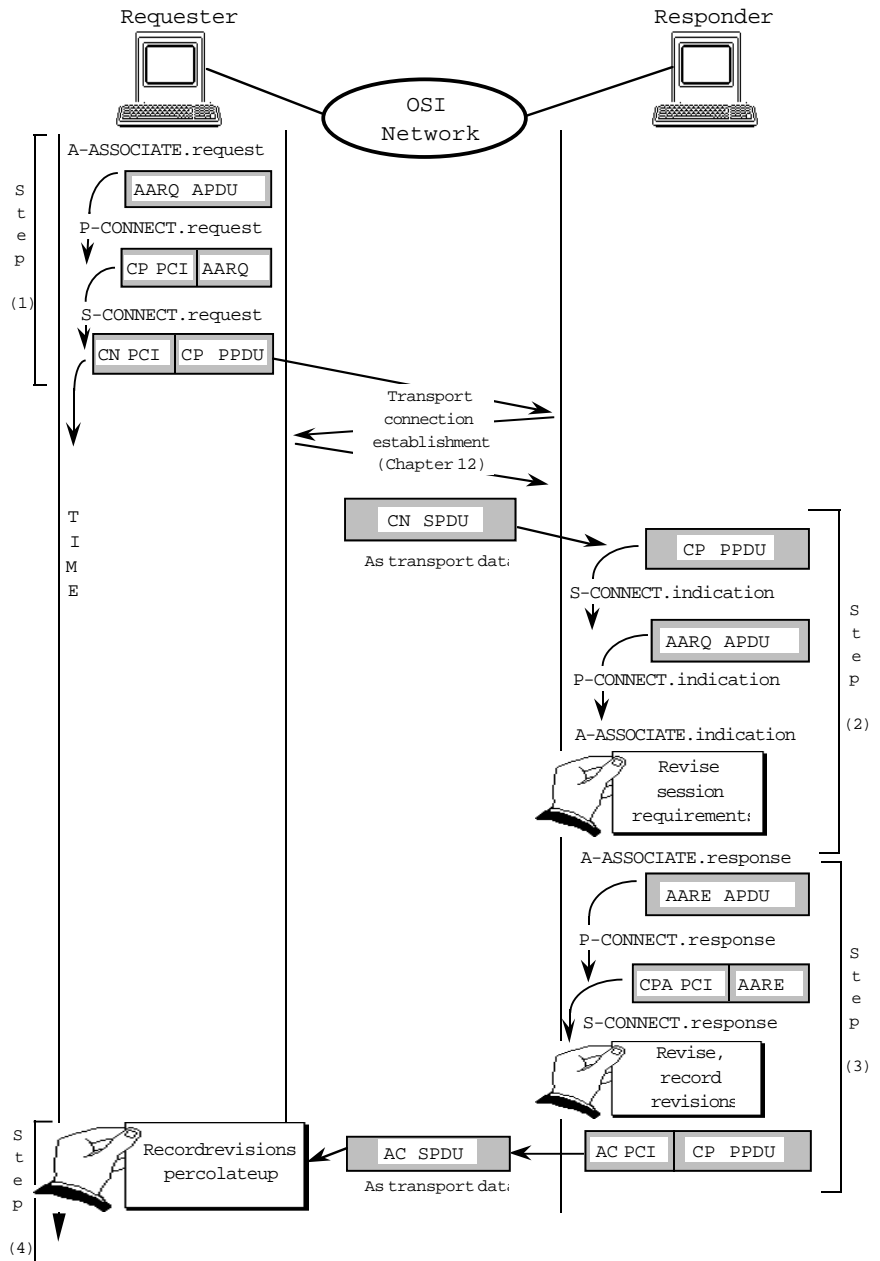


FIGURE 11.6 Negotiating User Session Requirements

the two communicating session service users to control the direction of information flow.

The notion of token-based data transfer is a pretty simple one: if you hold a token, you can use the service governed by that token (in this case, you can use the normal session data service); if you don't have the data token, you can request it using the please tokens service, or you can wait until the data token is yielded via the give tokens service, but you can't use the normal session data service until you hold the data token.

There are several ways to bypass the data token. In addition to the normal data transfer service, there is an *expedited data transfer* service. Expedited data are uninhibited by token/flow control but extremely limited in size (a mere 14 bytes). Originally, this data service was perceived to facilitate virtual terminal services, but outside of this, it's not especially useful, particularly since virtually nothing can be BER-encoded in 14 or fewer bytes. *Typed data transfer* can be used even if you don't hold the data token; it's sort of a "token be damned, I'm sending data" service. Of course, restrictions apply: the typed data service is available only when the half-duplex mode is selected. It is used by commitment, concurrency, and recovery (ISO/IEC 9804: 1990) and the Virtual Terminal (VT; ISO/IEC 9040: 1990). CCR uses typed data to request a restart, whereas Virtual Terminal uses typed data to switch VT profiles. Both CCR and VT use the half-duplex mode and clearly find occasions when it might have been wiser to use full-duplex, if only to have it available for exception cases. (This probably argues strongly for having standardized a full-duplex-only session service rather than a full-duplex service *plus* a half-duplex service with a patchwork exception service.)

A fourth data functional unit, *capability data*, is best understood, and only applicable, in the context of activity management.

Activity Management Functional Unit Chapter 10 discusses how the RT-TRANSFER service accepts arbitrarily large user data from a user element and treats each submission as a separate transfer activity (an individual application protocol data unit). At the session layer, the activity concept is preserved: APDUs submitted to the presentation layer in P-DATA.requests can be distinguished as individual activities by using the services that comprise the *activity management functional unit*. In particular, activity services are used to identify the beginning and end of an activity (S-ACTIVITY-START, S-ACTIVITY-END), to interrupt and later resume an activity (S-ACTIVITY-INTERRUPT, S-ACTIVITY-RESUME), and to discard an ongoing activity, (a transfer that for some reason has become expendable and can be trashed [S-ACTIVITY-DISCARD]).

Session activity management services are accessed by application entities using activity pass-through facilities of the presentation layer. The P-ACTIVITY-START and P-ACTIVITY-END primitives, for instance, map directly onto the S-ACTIVITY-START and S-ACTIVITY-END primitives. As an example, Figure 11.7 illustrates the sequence in which an activity is started by the application entity using the P-ACTIVITY-START primitive; the request is passed through to the session service in the form of a directly mapped S-ACTIVITY-START, which is communicated across the network via the activity start packet (in Figure 11.7, the AS SPDU).

If a single application entity is using a presentation connection, activities aren't very interesting. If the presentation connection is shared between two application entities between ASEs of the OSI Message Handling System and the Directory, for instance—activity management can play an interesting role. Consider, for example, a situation in which the MHS and the Directory share a presentation connection between two computers, “Michaelangelo” and “Donatello.” The message transfer application service element at Michaelangelo is in the process of forwarding a jumbo mailgram to Donatello when a directory user attempts to retrieve some naming/ addressing information (perhaps the name of a bodacious pizza parlor). Rather than delay the directory request until

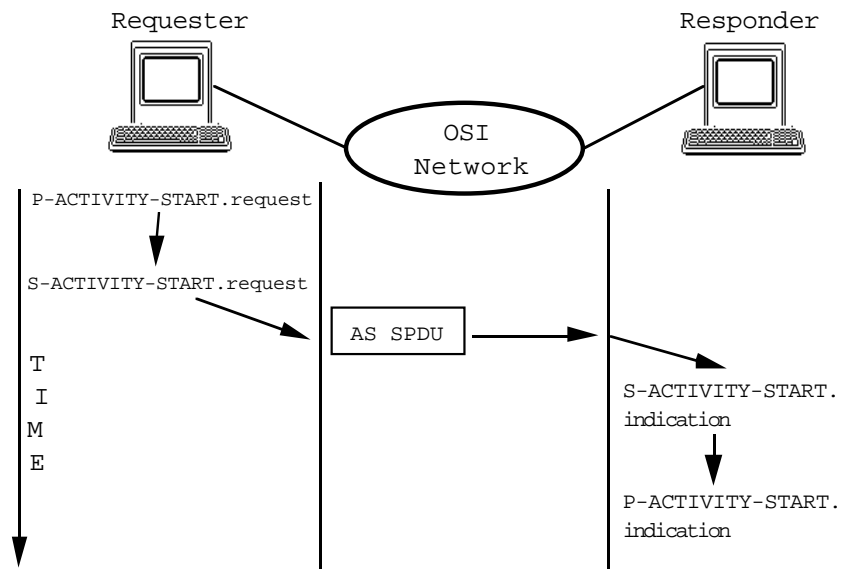


FIGURE 11.7 Activity “Pass-throughs”

the jumbo mailgram is transferred, the sending application entity can interrupt the mail activity using the S-ACTIVITY-INTERRUPT service (via the P-INTERRUPT pass-through). The progress of the mail activity is suspended, and a directory activity is started. When the directory activity is ended, the progress of the mail activity is resumed using the S-ACTIVITY-RESUME service (via the P-ACTIVITY-RESUME pass-through).

The S-ACTIVITY-INTERRUPT service may also be used in situations in which an application process is temporarily unable to continue processing the activity in progress (its ability to continue to receive data is compromised). In such cases, a new activity may not be started (a reason for interrupting the ongoing activity may be provided to the session layer via the pass-through mechanism; this is conveyed in session protocol, and no presentation protocol is involved).

Figure 11.8 illustrates the sequence in which an activity is interrupted. An activity “foo” is started (an AS SPDU is sent), data are transferred (via the normal data-transfer mode, DT SPDU), then “foo” is interrupted (in this case, to start an activity “bar”). The initiator sends an activity interrupt packet (AI SPDU), and the responder acknowledges the interrupt request using the activity interrupt acknowledgment packet (AIA SPDU). The initiator then starts activity bar, sends data, and resumes activity “foo” using the activity start, normal data and activity resume packets (AS, DT, and AR SPDUs, respectively).

Activity management is a token-based service; only the holder of the *major/activity* token may interrupt or start an activity.⁴ Whenever an activity is started, an *activity identifier* is assigned to that activity. If an ongoing activity is interrupted, the state of both the “old” and “new” activities is maintained. If, for example, major, minor, and/or resynchronization services are in use, the value(s) of the serial number(s) significant to the (old) activity at the point at which it was suspended is (are) stored for that activity, and the serial number(s) may be used for the new activity.

In case readers wondered whether the designers of the OSI session layer had overlooked *anything*, rest easy—they did not. Leaving absolutely nothing to chance, they even considered the scenario in which an

4. When transport expedited data service is available, a special SPDU, called a prepare (PR) SPDU, is used to “warn” the peer session provider that something “big” is about to happen—a major synchronization acknowledgment, resynchronize, or resynchronize acknowledgment packet (MHA, RS, or RA SPDU) is coming. The PR SPDU is sent on the transport expedited flow, indicating that incoming SPDUs received on the transport normal data flow may be discarded under certain circumstances. For example, when an activity is interrupted, a PR SPDU is used to signal “prepare to resynchronize,” letting the peer session provider know that it can ignore incoming normal data flow until the AI SPDU is received.

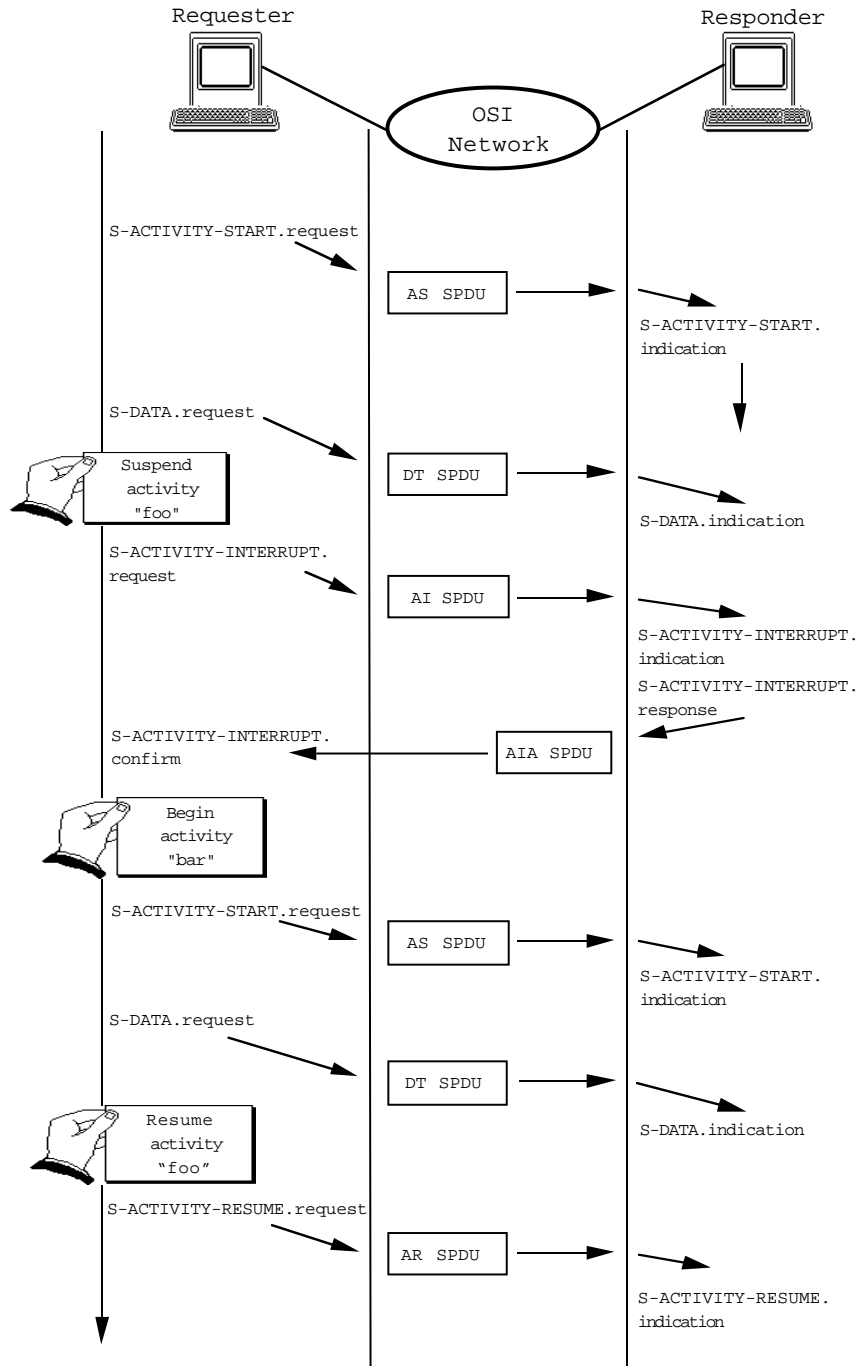


FIGURE 11.8 Interrupting an Ongoing Activity

activity has just ended, and before another activity is started, there might arise a need for an application to send a small amount of information—say, a nice, round number like 512 octets—prior to invoking the ACTIVITY-START service. (Hey, who knows? It could happen. Really.) *Capability data transfer* offers just such a service. It can be invoked only when activity services are available but no activity is in progress, and only if one holds the major/activity token and has the right to start the next activity (if available, one must also hold the data and/or minor synch token as well). Capability data transfer is sort of like a TV commercial: it can only be invoked “between” activities. Unlike the other data transfer services offered by the session layer, it is a confirmed service. This is necessary to ensure that the data transfer is completed before a new activity is started.

Major and Minor Synchronization Functional Units Chapter 10 introduced the concept of checkpointing, a facility that enables applications to identify during data exchange “points” in the transfer to which they may return in order to recover from (temporary) communications failures with a minimum amount of retransmission. The session *S-SYNC-MAJOR* and *S-SYNC-MINOR* services are used to provide the checkpointing, and the session *S-RESYNCHRONIZE* service is invoked to recover from temporary communication failure with a minimum of retransmission. As with the other pass-through services, these session services are indirectly accessed by application service elements via the corresponding presentation services (*P-SYNC-MAJOR*, *P-SYNC-MINOR*, and *P-RESYNCHRONIZE*, respectively).

All of the synchronization services use a *serial number*, a binary-coded decimal number between 0 and 999,999, to identify points in the byte stream. The decimal number bears no relationship to the octet position of the data being transferred; rather, it has a specific and somewhat different meaning or purpose for each synchronization service. In the major synchronization service, for example, the serial number indicated in an *S-SYNC-MAJOR* service invocation identifies both the end of a previous unit of communication called a *dialogue* and the beginning of the next. An important characteristic of a *dialogue unit* is that once an *S-SYNC-MAJOR.request* has been made, the requester can make no further service requests—especially additional data-transfer requests—of any kind until it receives the corresponding *S-MAJOR-SYNC.confirm*. The *S-SYNC-MAJOR.request* demands that no further action be taken with regard to transferring data within this activity⁵ until both the sender and

5. Note that if the activity functional unit is not selected, there is, in effect, one and only one activity transmitted over the session connection.

the receiver share a common understanding that previously transferred data have been secured (note that the current activity can be interrupted or discarded, or the session connection can be aborted).

The S-SYNC-MINOR service provides secondary or finer granularity to the checkpointing within an activity. Minor synchronization is distinguished from major synchronization in the following ways:

- An S-SYNC-MINOR.request may be issued as a confirmed or an unconfirmed service, at the sender's discretion.
- While an S-SYNC-MINOR.request is outstanding, any other service can be requested, including additional S-SYNC-MINOR.requests.
- The confirmation of an S-SYNC-MINOR.request or an S-SYNC-MAJOR.request implicitly confirms any outstanding S-SYNC-MINOR.requests.

Just as major synchronization points structure data exchange within an activity, minor synchronization points structure data within a dialogue unit. An example of a properly structured activity is illustrated in Figure 11.9.

Both the major and minor synchronization functional units include the give tokens and please tokens services; these are used to assign control of the *major/activity* token and *synchronize-minor* token. Only the session service user that holds the major/activity token may invoke the S-SYNC-MAJOR service; similarly, only the session service user that holds the synchronize-minor token may invoke the S-SYNC-MINOR service.

Symmetric Synchronization Some applications can operate in full-duplex mode. A set of computers that form a constantly replicating database, for example, could exchange information bidirectionally. In such scenarios, each application must have the ability to checkpoint both directions of information flow independently; the synchronization services should be symmetric. At the session layer, this translates into pro-

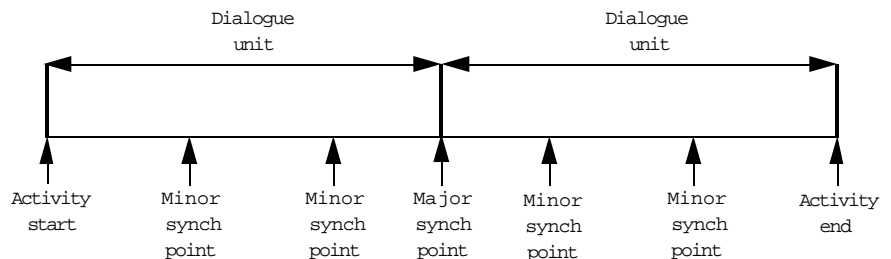


FIGURE 11.9 Structured Exchange of Data

viding a capability for both session service users to insert checkpoints into their “send” flow. When symmetric synchronization is used, a session entity maintains two serial numbers—one to checkpoint what it sends and one to checkpoint what it receives. ISO/IEC 8326 Addendum 1: 1987 and ISO/IEC 8327 Addendum 1: 1987 identify the extensions to the major and minor synchronization services to accommodate symmetric synchronization. Basically, two serial numbers are identified/ negotiated/exchanged at connect time and during activity start or re-sume. The values of both serial numbers are used to identify a major synchronization point or an activity end/interrupt. If resynchronization services are selected, resynchronization can be invoked independently on either direction of information flow or simultaneously (here, both serial numbers must be provided).



The original session service and protocol standards were designed to be compatible with the CCITT Recommendation T.62 session protocol for Teletex services, and the agreement between ISO and the CCITT was that the 1984 X.215/X.225 recommendations and the ISO/IEC 8326/8327 session standards would contain identical text. Now, T.62 session services were designed to operate in a half-duplex mode. Since only one of the session service users transferred data at a time, there was a need for only one session synchronization serial number. When full-duplex operations were introduced into the session standard, a second serial number was required to enable session service users to control each direction of information flow independently—to operate minor synchronization and resynchronization services. This is all well and good, but the timing of this “discovery” was unfortunate. Since CCITT closed the 1984 study period while symmetric synchronization was still under study, the Red Book versions of the CCITT recommendations and the ISO session standards were published without it. Symmetric synchronization services and protocol were introduced as addenda to the session service (ISO/IEC 8326 Addendum 1: 1987) and protocol (ISO/IEC 8327 Addendum 1: 1987) and in the 1988 Blue Book recommendations.

Resynchronize Functional Unit The S-RESYNCHRONIZE service is used to:

- Recover from temporary loss of a transport connection with a minimum of retransmission. Here, the *restart* option is indicated in the S-RESYNCHRONIZE.request and conveyed in the session resynchronize packets.
- Discard or *abandon* data hitherto associated with the current dialogue unit.

- *Set* the synchronization-point serial number to any valid value.

In all forms of the S-RESYNCHRONIZE service, the requester may propose a (new) assignment for the available tokens (or it may indicate that the accepter is allowed to assign token ownership). In the case of a “restart” request, the requester indicates the serial number at which the restart is to commence; it is effectively saying, “I’m confident that the transfer state is reliable to up to this point; let’s resume transfer here.” The accepter sets the lowest serial number to which a synchronization point confirmation is expected; it also sets the next serial number expected to the restart serial number indicated in the session resynchronize (restart) packet (RS-r SPDU). At this point, the accepter has modified its state machine to reflect that of the requester, so it sends a resynchronize (restart) acknowledgment packet (the RA-r SPDU). Upon receiving the RA-r SPDU, the requester may retransmit data (see Figure 11.10). The accepter may discard any data secured and associated with a serial number greater than the serial number indicated as the restart serial number; from the requester’s perspective, these are “suspect” or unconfirmed data.

In the case of an “abandon” request, the requester does not send a serial number, since it’s effectively saying, “Let’s start over.” In response to the session resynchronize (abandon) packet (the RS-a SPDU), the accepter returns a session resynchronize (abandon) acknowledgment packet (the RA-a SPDU), discards all data of this dialogue unit that were previously secured, and awaits “new” transfer. In this case, the accepter expects to proceed as if resynchronization hadn’t happened at all. The serial number indicated in the next major/minor synchronization request will be the “next expected serial number” indicated in the RS-a SPDU, and the lowest serial number to which a resynchronization restart may be set becomes 0.

In the case of a “set” request, the requester selects any valid serial number and sends this in an RS-s SPDU. The accepter sets the next expected serial number to the value received in the RS-s SPDU, and returns an RA-s SPDU. Following receipt of the RA-s SPDU and confirmation of the S-RESYNCHRONIZE.request, the requester will use the “set value” in subsequent synchronization requests, and the lowest serial number to which a resynchronization restart may be set becomes 0.

Like the major and minor synchronization services, application entities access the session resynchronization service via a pass-through service of the presentation layer (P-RESYNCHRONIZE).

Exceptions Functional Unit Considering how complicated the session protocol can be, it seems only fitting that a service was designed to allow

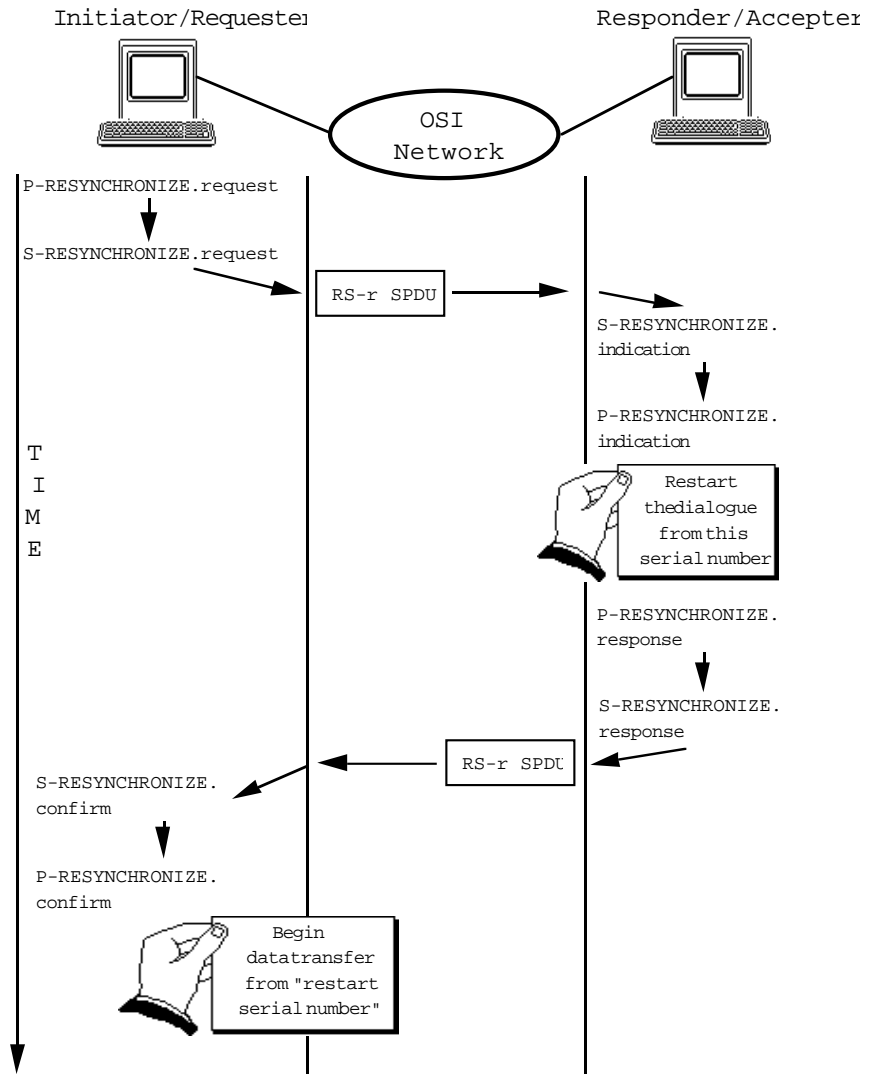


FIGURE 11.10 Resynchronization "Restart"

the session service provider to notify session service users of "unanticipated situations not covered by other services" (from ISO/IEC 8326: 1987). The presentation exception reporting service (S-P-EXCEPTION-REPORT) can be used to signal a protocol or "nonspecific" error to session service users. If a protocol error has been detected by either session protocol machine (SPM), that SPM may attempt to transfer an exception report packet (an ER SPDU), which contains as a parameter the SPDU

that the session protocol machine has identified as objectionable. Following an indication of this sort, the session service users are encouraged to resynchronize, interrupt or discard the current activity, or abort the session connection. (Note that only if the transfer and interpretation of the ER SPDU are successful will both session service users receive an indication that an exception condition has been detected.)

An exception reporting service is also available to the session service users. If a session service user encounters an exception condition, it can issue an S-U-EXCEPTION.request. The local session protocol machine will attempt to transfer an exception data packet (an ED SPDU) to the remote SPM containing an indication of the sort of physical, emotional, or psychological problem the requester is experiencing. If the ED SPDU arrives and can be processed, an S-U-EXCEPTION.indication may be generated.



Although it is possible to attempt to resynchronize, interrupt, or discard the current activity, or even yield tokens to a session service user that thinks it can solve the problem if only it could send data, it's generally a good idea for the session service user to follow exception service requests with an abort. Dynamic, self-recovering protocol implementations are hard to find.

Negotiated Release Functional Unit The negotiated release functional unit consists of orderly release and the give tokens and please tokens services. The *orderly release* service (S-RELEASE) resembles the graceful close offered by TCP (see Chapter 12); both session service users cooperate to ensure that all data in transit have been delivered (acknowledged) before the session connection is closed. Orderly release is influenced by the *release token*; if an S-RELEASE.indication arrives and the release token is available, the acceptor may refuse to release the connection (once again belying the term *accepter*). This situation might occur when a mail application “foo” has nothing more to send to its peer “bar” and wishes to release the association, but “bar” still has messages to forward to “foo”.

Orderly release is initiated using the S-RELEASE.request service primitive. A session finish packet (an FN SPDU) is sent to the acceptor; if the acceptor agrees to close the session connection, it returns a session disconnect packet (DN SPDU), confirming the S-RELEASE. A *transport disconnect* parameter in the DN SPDU may be used to indicate whether the transport connection hitherto supporting this session connection is to be reused, and the session protocol machine will proceed according to

what is indicated. If the accepter wishes to maintain the session connection, it responds to the FN SPDU with, appropriately, a session not finished packet (NF SPDU), resulting in a rejection of the S-RELEASE service. This is permitted only if the negotiated release functional unit was accepted when the session was established.

Abort Services The *session abort* service (S-U-ABORT) of the kernel functional unit supports the abort services of the association control service element and the presentation layer. What more can one say? An application entity has observed that things have gone to hell in a hand-basket and indicates that it wants to tear down the connection. The *presentation abort* service (S-P-ABORT) is used to indicate that something disastrous has occurred in or below the session layer; the protocol implementation is broken, for example, or the transport connection has disconnected. If the latter is true, the session layer initiates an S-P-ABORT indication to notify the presentation layer.

Session Protocol

Most of the descriptions of the session protocol have been vicious and unkind. Regrettably, they are deserved, and no amount of historical perspective can alter the fact that the session protocol, from encoding to operation, is neither elegant nor efficient.⁶ In version 1, there are 29 states in the protocol machine, further complicated by 75 predicates—and this doesn't take into account additions to accommodate unlimited user data (ISO/IEC 8327 Addendum 2: 1987) or symmetric synchronization (ISO/IEC 8327 Addendum 1: 1987). The encoding of the session protocol itself is a "fixed-field format" lover's worst nightmare. For starters, all session protocol data units have the following components:

SPDU identifier (SI)	A single octet; identifies the type of SPDU
Length indicator (LI)	A single octet; specifies the length of the SPDU in octets
Parameter identifier (PI)	An individual or a group parameter (variable-length)
User information	(Variable-length)

6. Upon first examination, it almost appears that its creators were a divided camp, half insisting on a "TLV" approach similar to ASN.1, the other half insisting on specification by the bits and bytes "as we've always done it in the lower layers." The true reason is not nearly this simple. The session protocol is arguably the worst example of design by committee; faced with multiple base documents to consider (one from ECMA [ECMA75], one from CCITT [CCITT T.62], others too embarrassing to mention), confronted with a commitment to align OSI and the CCITT Teletex protocols, and expected to come to closure on a standard prior to the completion of the 1984 CCITT study period, the committee collapsed under pressure and, "in the spirit of compromise," adopted a combination (not quite the union) of the services of the documents under consideration. At the service level, this was a sizable enough pill to swallow; the ramifications to the protocol (of course, everyone insisted on perpetuating their own bits as well) were nearly ruinous.

The *parameter identifier* (PI) specifies either an individual parameter or a parameter group. Individual parameters have the following components:

Parameter identifier	A single octet; identifies the parameter
Length indicator	A single octet; specifies the length of the parameter in octets
Parameter value	The value assigned to the parameter

whereas *parameter groups* are sets of logically related individual parameters:

Parameter group identifier (PGI)	A single octet; identifies the parameter group
Length indicator (LI)	A single octet; specifies the length of the parameter group in octets
PI ₁	Identifier of first parameter in group
LI ₁	Length of first parameter in group
PV ₁	Value assigned to first parameter in group
•	
•	
•	
PI _n	Identifier of <i>n</i> th parameter in group
LI _n	Length of <i>n</i> th parameter in group
PV _n	Value assigned to <i>n</i> th parameter in group

This encoding style is decidedly complex; the composition of the sample session connect packet (CN SPDU) in Figure 11.11 illustrates how quickly the protocol header becomes littered with “nested” parameter identifiers and length indicators (LI).

Octet No.	CN SPDU	Meaning/Significance	AC SPDU	Octet No.
0	SI = 13	SPDU identifier for connect SPDU	SI = 13	0
1	LI = 723	Total length of SPDU (does not include the SI or LI)	LI = 726	1
2	PGI = 1	Connection identifiers parameter group identifier	PGI = 1	2
3	LI = 138	Length of connection identifiers parameter group	LI = 138	3
4	PI = 10	Calling (CN) or called (AC) SS user reference	PI = 9	4
5	LI = 64	Length of SS user reference (0 < <i>n</i> ≤ 64 octets)	LI = 64	5
6–69	<Value>		<Value>	6–69
70	PI = 11	Common reference parameter	PI = 11	70
71	LI = 64	Length of common reference (here, 64 octets)	LI = 64	71
72–135	<Value>		<Value>	72–135
136	PI = 12	Additional reference information parameter	PI = 12	136
137	LI = 4	Length of additional reference information	LI = 4	137
138–141	<Value>		<Value>	138–141
142	PGI = 5	Parameter group identifier for connect/accept item	PGI = 5	142

143	LI = 64	Length of connect/accept item parameter group	LI = 64	143
144	PI = 19	Protocol options parameter	PI = 19	144
145	LI = 1	Length of protocol options parameter	LI = 1	145
146	<Value>	Ability to receive extended concatenated SPDUs	<Value>	146
147	PI = 21	Transport service data unit (TSDU) maximum-size parameter	PI = 21	147
148	LI = 4	Length of TSDU maximum-size parameter	LI = 4	148
149–152	<Value>	Maximum size of TSDUs (for both directions)	<Value>	149–152
153	PI = 22	Version number	PI = 22	153
154	LI = 1	Length of version number	LI = 1	154
155	<Value>		<Value>	155
156	PI = 23	Initial serial number parameter	PI = 23	156
157	LI = 6	Length of initial serial number parameter	LI = 6	157
158–163	<Value>	A BCD-encoded value between 0 and 999,999	<Value>	158–163
164	PI = 26	Token-setting item parameter	PI = 26	164
165	LI = 1	Length of token-setting item	LI = 1	165
166	<Value>	Four “bit pairs” representing initial holder of token	<Value>	166
	n/a	Token item parameter	PI = 16	167
	n/a	Length of token item	LI = 1	168
	n/a	Indication of which tokens are requested by caller	<Value>	169
167	PI = 20	Session user requirements parameter	PI = 20	170
168	LI = 2	Length of session user requirements parameter	LI = 2	171
169–170	<Value>	Each bit represents a session functional unit requested by SS user	<Value>	172–173
171	PI = 51	Calling session selector parameter	PI = 51	174
172	LI = 16	Calling session selector length ($0 < n \leq 16$ octets)	LI = 16	175
173–188	<Value>		<Value>	176–191
189	PI = 52	Called (responding) session selector parameter	PI = 52	192
190	LI = 16	Session selector length ($0 < n \leq 16$ octets)	LI = 16	193
191–206	<Value>		<Value>	194–209
207	PI = 193	User data parameter	PI = 193	210
208–210	LI = 512*	Length of user data parameter	LI = 512	211–213
211–722	<Value>	User data—contains CP PPDUs, AARE APDU	<Value>	214–725

* When the length indicator value is less than 256, a single octet is used; when the value is greater than 256, 3 octets are used, with the first octet set to binary 1s to indicate that the following 2 octets contain a length between 0 and 65,535.

FIGURE 11.11 Encoding of CN and AC SPDUs

If readers see a potential “chicken and egg” situation regarding the negotiation of maximum transport service data unit size in session connection establishment here, they are to be commended. A maximum of 32 octets of transport user data may be transferred via the T-CONNECT primitives (see Chapter 12); it is thus necessary to first establish a transport connection before the session connect packet can be sent. Now, although the maximum length of transport user data that may be transferred over a transport connection is theoretically unbounded, it’s generally a good idea to negotiate the largest possible transport data unit size

during transport connection establishment to minimize fragmentation/reassembly, and it would be convenient indeed if the session layer would notify the transport layer of the maximum TSDU size when it requests a transport connection. Unfortunately, the primitives of the T-CONNECT service don't offer a maximum TSDU size parameter, so indication of this useful bit of information is a *local implementation matter*.⁷

A Word about Concatenation Another truly confusing aspect of the session protocol is the notion of concatenation of session packets (SPDUs) into TSDUs. SPDUs are categorized into three groups:

1. Category-0 SPDUs (give and please tokens SPDUs) are treated as responsible adults; they may be mapped one-to-one onto a TSDU or concatenated with another (category-2) SPDU.
2. Category-1 SPDUs (connect, accept, refuse, finish, disconnect, not finished, give tokens ack, give tokens confirm, exception data, typed data, abort, and prepare SPDUs) are treated as outcasts and must be mapped one-to-one onto TSDUs.
3. Category-2 SPDUs (data transfer, the major/minor/resynchronize, activity, capability, and exception SPDUs) are treated as small children and must be "accompanied by an adult," or category-0 SPDU.

A category-0 SPDU is always the first session packet in a transport service data unit. If basic concatenation is used, a second session packet may be appended to the first if it comes from the set of category-2 SPDUs, and if *extended concatenation* is used, you can piggyback multiple category-2 SPDUs. Here, the rules are so complicated that the session protocol doesn't even attempt to describe them in text; it merely provides a table. Basically, follow these precedence rules: activity management packets precede major/minor synchronization packets, which precede data transfer packets.

Data Transfer Normal data transfer packets always accompany (at least) a give or please tokens packet. An example of the simplest form of encapsulation is provided in Figure 11.12.

Of course, if a session protocol implementation is *accomplished*, it will be able to parse and process SPDUs concatenated in TSDUs as complex as those illustrated in Figure 11.13.

As a final example, Figure 11.14 illustrates a simple sequence of

7. ISO standards internationally leave out implementation details. This is so orthogonal to the way Internet Requests for Comments are written that one astonished Internetter suggested, "The sum of what falls under 'local implementation matter' in ISO standards could fill an ocean."

Octet No.	SPDUs in TSDU	Meaning/Significance
1	SI = 1	Give tokens SPDU identifier
2	LI = 3	Total length of give tokens parameters
3	PI = 16	Token item parameter identifier
4	LI = 1	Length of token item
5	<Value>	Indicates which tokens are being given by sending SS user
6	SI = 1	Data-transfer (DT) SPDU identifier
7	LI = 255	Indicates that next 2 octets contain “the real LI”
8, 9	LI = 2,001	Length of the DT SPDU
10	PI = 25	Enclosure item parameter identifier
11	<Value octet 1>	Indicates that this is the beginning/middle/end of SPDU
12–2,011	<Value octets 2–2,001>	User information

FIGURE 11.12 Encapsulation of SPDUs—Simplest Form

session primitives and packets for a session connection in which the activity management, major synchronization, and minor synchronization functional units are used in a half-duplex mode of operation; expedited data transfer is not available, and the exception services were selected but not required. In the example, the requester is assigned all the tokens—major/activity, synchronize-minor, release, data—at connection-establishment time. Also, for illustrative purposes, both confirmed and unconfirmed minor synchronization are shown.

In this example, an activity composed of two dialogue units was transferred from the requester to the acceptor. Although quite simple, the exchange represents the way the session service might be used by the OSI Message Handling System to transfer a single mailgram from one mail server to another.



Rose (1990) calls attention to many of the flaws and weaknesses of the session protocol. His criticism is scathing and complete, and there is little need here to beat a dead horse. To Marshall's credit, however, he didn't just sit on his hands and poke fun; he put together a complete implementation of the session layer in the ISODE, which is probably the most widely used upper-layers implementation in deployment today. Some OSI folks view Marshall as an enemy of the state; considering how much the availability of the ISODE has contributed to OSI deployment today, they might consider looking past the bluster and hype and appreciate all the good his OSI implementation has done.

Final word on the session protocol: it's ugly, but then, so are the UNIX awk command, the Report Program Generator Language (RPG), and dozens of

Give tokens SPDU	Activity-start SPDU	Major synch SPDU	
Give tokens SPDU	Activity-start SPDU	Minor synch SPDU	Data-transfer SPDU

FIGURE 11.13 Example of SPDUs Concatenated in TSDUs

other technologies. The only thing that matters is whether it helps people do useful work . . .

Putting It All Together

When OSI is presented in a “layer-by-layer” manner, one is inclined to conclude that there are lots of connections, perhaps altogether too many! This perception can be partially attributed to the rigid, formal manner in which layers are defined: after all, seven separate layers must have seven separate connections, right? Well, not necessarily so.

The risk of examining OSI one layer at a time is that interdependencies between layers are often obscured. When a “bottom-up” approach to understanding and interpreting OSI is applied as well—from the physical layer to the application layer—the problem is exacerbated. A bottom-up approach begins by showing how the physical layer schlepps electrons across physical connections and enables carriage of data-link bits through materials like copper and glass, and proceeds to explain how data-link connections recover the unfortunate bits that get smashed along the way. This is followed by explanations of how datagrams are forwarded (temporary respite) or how network connections are set up between computers. Then, a transport connection is established for reliability. Next, a session connection. . . . Soon, you are gently lulled into a rhythm of “another layer, another connection.”

Above the Internet layer in TCP/IP, there are only two connections and, with them, two associated “state machines”: an “application” connection and an end-to-end connection (transport). An Internet mail application (RFC 822/SMTP mail; see Chapter 8)—the UNIX sendmail command, for example—establishes “mail” connections to other mail applications to forward, deliver, and receive preformatted mail messages. In a UNIX environment, sendmail resides in user space; it operates over TCP connections accessed via the UNIX socket().

In OSI, the same two connections exist: an MHS-based mail application establishes an association for messaging and runs this over a transport connection. Although the situation is somewhat obscured by

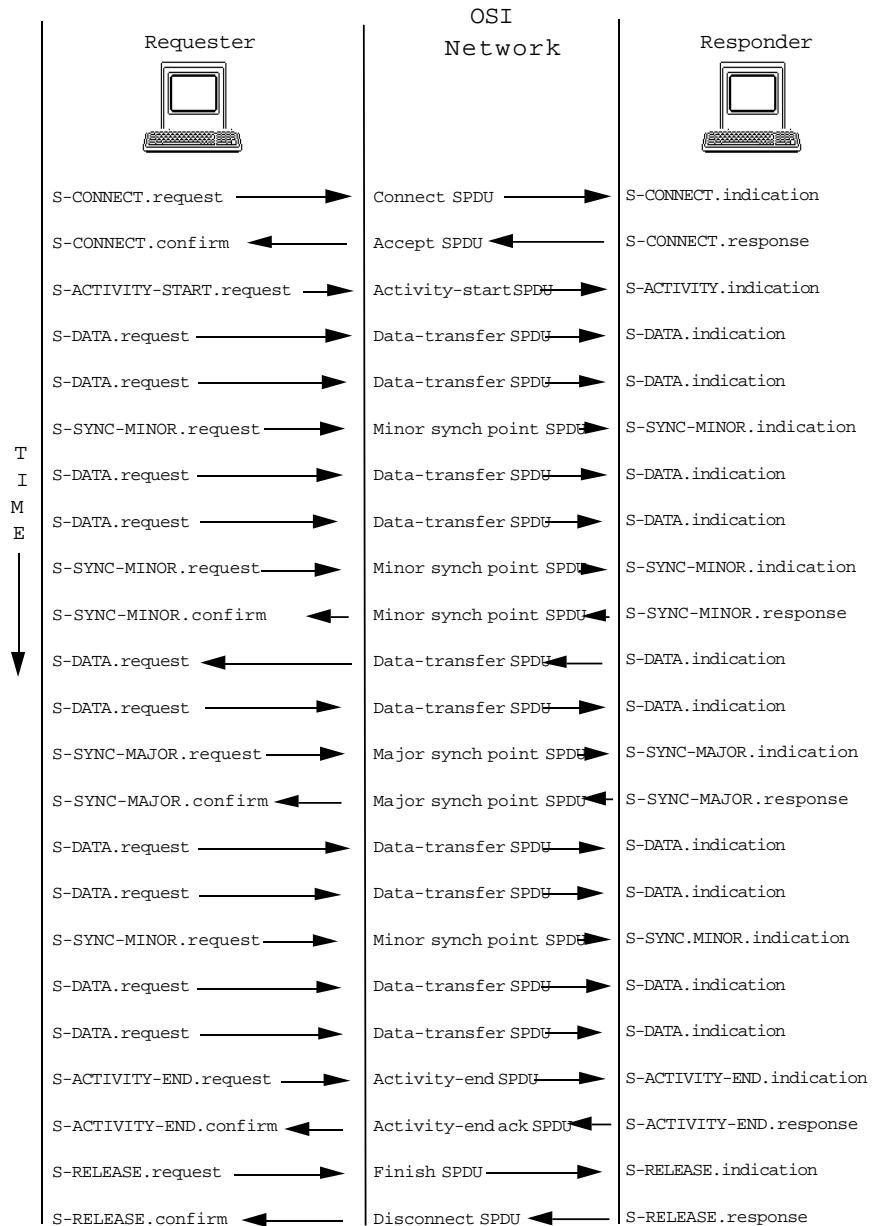


FIGURE 11.14 Sample "Session"

the formal "layer at a time" presentation, close examination of the interaction among the application, presentation, and session service elements

reveals that a single connection state rather than three exists for each end-user application. Association control and presentation connection protocol control information should be piggybacked on session connect and accept packets, effectively creating a pseudoheader for upper layer association management (see Figure 11.15); one software process rather than several can then be invoked to establish or accept an association. (Note: A software process is represented graphically in Figure 11.15 by a round-edged rectangle, itself containing subprocesses or routines similarly represented.) In a UNIX environment, the entire OSI upper layers can be placed in user space, and end-to-end transport can again be accessed via the `socket()`.

In the figure, the fictitious command “Connect (*x, y, z*)” is used to establish an association between this mail application and another. Once the association is established, control is returned to the application software, which proceeds to send or receive data through some software process responsible for managing data transfer; here, the fictitious operators “Send()” and “Recv()” are invoked to exchange messages. (Note that the association management software process will “sleep” until it is called upon to terminate the association or handle errors.)

Under the direction of the data transfer manager, a mail message is transferred as an activity within the association. Session layer “jimmies”⁸ like major and minor synchronization may be used by the mail application if the message is large and recovery from temporary loss of the presentation connection with minimal retransmission is desirable. Semantically, the state of the application service elements is tightly coupled to the session connection when services such as these are invoked (you know the drill: reliable transfer service says, “Do that funky synch thing” to presentation, which echoes it to session, while MHS waits patiently for a confirmation before proceeding to the next activity); syntactically, this merely introduces additional bits in the information streamed across the single end-user connection.

The ISODE (Kille and Robbins 1991) is an example of how one might implement the OSI upper layers as a set of C libraries consisting of the “core ASEs,” presentation, and session that are loaded with end-user application programs such as the OSI Directory, Message Handling System, and FTAM, which can then be run over TCP or any equivalently featured end-to-end transport using a transport convergence protocol

8. Jimmies are ant-shaped bits of candy sprinkled over ice cream—i.e., add-ons to an already tasty treat. Here, the term is used to indicate that extra degrees of control can be exercised over information exchange by invoking session services.

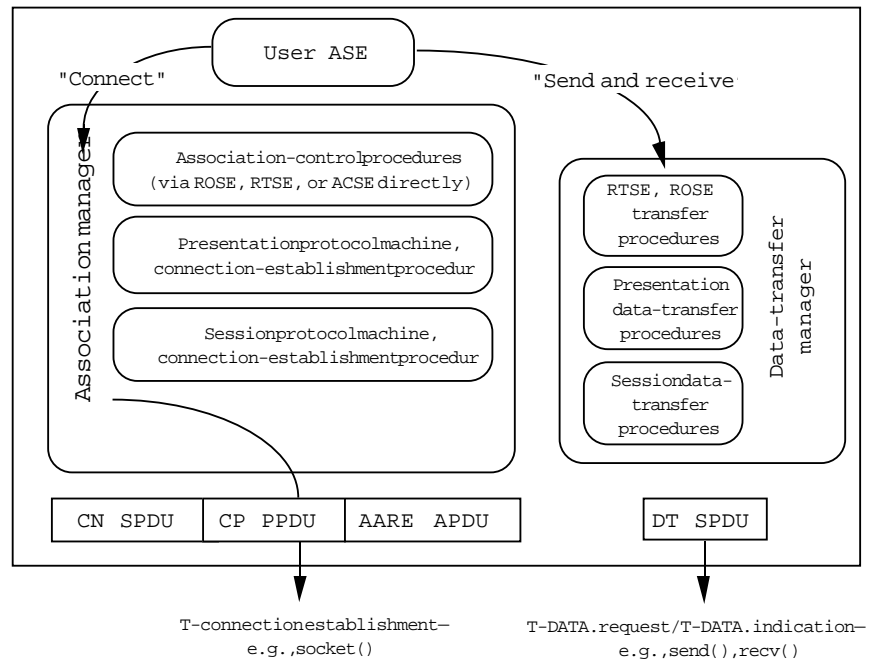


FIGURE 11.15 Association Management

such as RFC 1006. (In the Wisconsin ARGO 1.0 and 4.3 BSD RENO UNIX kernels, an end-to-end OSI transport connection is accessed via sockets as well, using extensions to this interprocess communication mechanism that accommodate a “sequenced-packet delivery”; see Chapter 12).

The Future of OSI Upper Layers

Since 1983, experts have claimed that (1) the organization of the OSI upper layers as described in the OSI reference model is a mess, and (2) the subsequent reconsideration of the application layer architecture (described in Chapter 6) yielded a structure that was more promising. Recently, ISO has extended the application layer structure to allow a single control function (CF) to supervise a set of application service elements, and a revision of the entire upper layer architecture is under consideration, which will essentially allow implementations to slice the upper layers “vertically” and may ultimately collapse the upper layers into a single, object-oriented “service layer” (Day 1992).

The *extended application layer structure* (XALS) and revised OSI upper layer architecture under study in ISO define *application service objects* (ASOs) that will contain multiple application service elements, some of these formed by grouping session functional units into application service elements and eliminating the session layer entirely. The existing presentation layer functionality will be subsumed within a new association control service element, which will offer an A-DATA service, and the presentation layer will be removed from the OSI reference model as well. The extended A²CSE (ASO association control service element) will then be bolted directly on top of the transport layer.

These changes affect a number of OSI standards, including the reference model, and they won't happen overnight. The current wisdom/optimism is that these moves will resolve some of the frequently criticized upper layer implementation difficulties. OSI upper layer implementations are criticized for having considerable overhead. The ISODE 7.0 implementation, for example, binds the entire session service library to each application process at run time; a possible result of rearchitecting the upper layer structure might be that future OSI-based programs/processes would be smaller, since the new ASOs would have only a subset of session functional units (basic synchronization and basic combined subsets rise from the ashes!).



There is, of course, a downside to these proposals; namely, the tremendous impact they would have on the few daring vendors that have ventured forth into OSI upper layers product deployment. Since, for reasons of efficiency already noted, OSI upper layer implementations tend to be closely coupled with the application service element, reorganization and reimplementation will be painful. Also, eliminating a layer of protocol header will wreak havoc on interoperability—the original and continuing goal of OSI, remember?—and require an extensive migration and coexistence plan. The moral? Prototype and measure, until a truly worthwhile “skinny-stack” approach emerges that supports a business case for transition from existing OSI upper layers to the new world.

Conclusion

This chapter concludes the discussion of the OSI upper layers. Having first examined application services at a conceptual and “features” level in Chapters 7, 8, and 9 (introducing directories, e-mail, and network management), the authors then discussed the service elements on which

distributed applications rely to provide such end services—in particular, application connection establishment, remote operations, and reliable transfer. This chapter has demonstrated how certain functions that application service elements provide to user elements or “specific” application service elements—the checkpointing, turn management, and activity management services that constitute reliable transfer—are performed at the session layer and how these are accessed via “pass-through” services provided by the presentation layer. Having completed a layer-by-layer description of the OSI upper layers, the authors “put it all together” and discussed how the existence of seven layers is not a mandate for (at least) seven connections—the interdependencies between the OSI upper layers may be exploited in prudent implementations, and everything in OSI “from session and above” can be implemented as a single connection. The chapter concludes with some insights into how implementation experience and hindsight may contribute to future refinements in the OSI upper layer structure.