10 "CORE" APPLICATION SERVICE ELEMENTS

Open any toolbox—irrespective of whether it belongs to a plumber, a carpenter, an electrician, or an automobile mechanic—and you will find a common, or "core," set of tools: subtleties aside, a pair of pliers, a screwdriver, and a wrench are useful and necessary tools, whether the application is plumbing, carpentry, electrical work, or automotive repair. In general, distributed applications also rely on a set of core capabilities; i.e., service elements that provide:

- The ability to initiate and terminate communications across a network (in this case, the OSI networking environment) and to ascertain prior to attempting to transmit data that the called application has all the facilities and capabilities required to interpret and operate on the data about to be sent. In OSI, the *association control service element* (ACSE) provides these capabilities.
- The ability to structure conversations between distributed applications (for example, providing the ability for applications operating over a full-duplex connection to "take turns") and the ability for an application to recover from disruption of underlying communications services without loss of data (for example, providing the ability for peer applications to indicate "Yes, I heard what you said up to the point where we were disconnected, and I've committed it to memory [written it down]"). In OSI, the *reliable transfer service element* (RTSE) provides application access to the complement of dialogue control capabilities offered in OSI: activity and turn management, synchronization, and resynchronization.
- The ability to perform functions at remote computers (remote oper-

ations, remote procedure calls). In OSI, the *remote operations service element* (ROSE) enables an application operating at one computer to request or direct an application operating at a second computer (or multiple computers) to share part of the processing burden. Conceptually, a remote operation might say, "Do this for me and inform me of the result when you've completed the task," or "Do this for me while I go off and do something equally important. Don't interrupt me unless something goes wrong."

These application service elements are the nucleus of an OSI application tool kit. (Readers are left with the exercise of determining which application service element corresponds to the pair of pliers, the screwdriver, and the wrench.)

Identifying only these application service elements as "core" elements is admittedly subjective. OSI tried to do so formally in the early stages of application layer standardization and failed because no consensus could be reached. In theory, the set of basic application service elements is open-ended; many others—notably, commitment, concurrency, and recovery (CCR); remote database access (RDA); and message handling—might eventually be included. Association control, reliable transfer, and remote operations, however, are likely to be included in nearly everyone's view; the treatment of "core" application service elements in this chapter, then, represents a minimalist's view.

Association-Control Service Element

The association control service element is included in all application context specifications[1] and is used to form a connection (called an *association*) between application entities. In fact, the *facilitating* aspects of establishing an association—notably, the exchange of naming information (especially application entity titles) and negotiation of the facilities that the communicating applications will need in order to communicate—are ACSE's primary role.

It is useful to continue with the analogy of association control's playing the role of a facilitator in OSI application-to-application commu-

1. Technically speaking, the ACSE service definition *strongly suggests* this, stating that "it is expected that the ACSE will be included in all application context specifications" (ISO/IEC 8649: 1988). The OSI *application layer structure* standard (ISO/IEC 9545: 1989), written a year later, states that the ACSE "is a necessary part of AEs." The reason for this ambiguity is revealed in a later discussion of *modes* of ACSE operation.

nications. In creating an application association, the association control service element truly facilitates a conference session; it identifies:

- Who will be talking (which application entities)
- What subjects will be discussed (which application service elements)
- The language(s) that will be spoken during the conference session (what abstract syntaxes will be exchanged)
- The "props" required for the conference session (what the presentation and session connections should look like)

Once it has established an association between two or more application entities, the association control service element fades into the background, reappearing only to assist in closing the conference session (and in extreme cases, to interrupt or cancel it).

The ACSE Service and Protocol

The formal mechanisms for establishing an application association are described in a service definition (ISO/IEC 8649: 1988) and a protocol specificaiton (ISO/IEC 8650: 1988). There are four association control services: *A-ASSOCIATE* provides "connection" establishment, *A-RELEASE* provides an orderly "disconnect," and the *A-P-ABORT* and *A-U-ABORT* services offer provider- and user-initiated disruptive "disconnects." Formally speaking, in establishing an association, 30 parameters are defined for the A-ASSOCIATE service. Simplified greatly, application entities use the parameters of the A-ASSOCIATE service primitives to identify:

- Themselves
- The application context required for this association
- The presentation context required for this association
- Presentation service requirements
- Session service requirements

(The complete set of A-ASSOCIATE service parameters is presented in Table 10.1.)

To invoke the service, an association control user (a.k.a. the requester) issues an *A-ASSOCIATE.request* primitive. The requester may be another standard application service element or a user element. In the *normal mode of ACSE operation*, association control uses some of the parameters passed in this request to construct the A-ASSOCIATE.request packet (in Figure 10.1, the AARQ APDU); most of the parameters conveyed in the AARQ APDU are identifiers. Application entities identify themselves through the exchange of their names—their *application entity*

TABLE 10.1 A-ASSOCIATE Service Parameters

Parameter Name	*Request*	*Indication*	*Response*	*Confirm*
Mode	User option	Mandatory(=)		
Application-context name*	Mandatory	Mandatory(=)	Mandatory	Mandatory(=)
Calling AP title*	User option	Conditional(=)		
Calling AE qualifier*	User option	Conditional(=)		
Calling AP invocation identifier*	User option	Conditional(=)		
Calling AE invocation identifier*	User option	Conditional(=)		
Called AP title*	User option	Conditional(=)		
Called AE qualifier*	User option	Conditional(=)		
Called AP invocation identifier*	User option	Conditional(=)		
Called AE invocation identifier*	User option	Conditional(=)		
Responding AP title*			User option	Conditional(=)
Responding AE qualifier*			User option	Conditional(=)
Responding AP invocation identifier*			User option	Conditional(=)
Responding AE invocation identifier*			User option	Conditional(=)
User information	User option	Conditional(=)	User option	Conditional(=)
Result			Mandatory	Mandatory(=)
Result source				Mandatory
Diagnostic*			User option	Conditional(=)
Calling presentation address	See ISO 8822	See ISO 8822		
Called presentation address	See ISO 8822	See ISO 8822		
Responding presentation address			See ISO 8822	See ISO 8822
Presentation context definition list*	See ISO 8822	See ISO 8822		
P-context definition result list*		See ISO 8822	See ISO 8822	See ISO 8822
Default presentation context name*	See ISO 8822	See ISO 8822		
Default presentation-context result*			See ISO 8822	See ISO 8822
Quality of service	See ISO 8822	See ISO 8822	See ISO 8822	See ISO 8822
Presentation requirements*	See ISO 8822	See ISO 8822	See ISO 8822	See ISO 8822
Session requirements	See ISO 8822	See ISO 8822	See ISO 8822	See ISO 8822
Initial synch point serial number	See ISO 8822	See ISO 8822	See ISO 8822	See ISO 8822
Initial assignment of tokens	See ISO 8822	See ISO 8822	See ISO 8822	See ISO 8822
Session connection identifier	See ISO 8822	See ISO 8822	See ISO 8822	See ISO 8822

*Not used in X.410-1984 mode of operation.
Note: In this table, the notation "(=)" is used to indicate that the value of the parameter (when it is present) in an indication or confirm primitive must be the same as the value of the same parameter in the corresponding request or response primitive (respectively). The notation "See ISO 8822" is used to indicate that the requirements for the parameter are specified in the service definition for the presentation layer (ISO/IEC 8822: 1988; see Chapter 11).

titles—which are defined as having two components: an *AP title* plus an *AE qualifier* (see "Names," in Chapter 5). Application entities further qualify this identification with *AE invocation identifiers,* which distinguish application entities at the process or run-time level.

```
AARQ-apdu ::=                  [APPLICATION 0] SEQUENCE
{ protocol-version                   [0] IMPLICIT BIT STRING
                                         { version1 (0) }
                                         DEFAULT version1,
application-context-name             [1] Application-context-name,

called-AP-Title                      [2]  AP-title
                                         OPTIONAL,
called-AE-Qualifier                  [3]  AE-qualifier
                                         OPTIONAL,
called-AP-Invocation-identifier      [4]  AE-invocation-identifier
                                         OPTIONAL,
called-AE-Invocation-identifier      [5]  AE-invocation-identifier
                                         OPTIONAL,
calling-AP-Title                     [6]  AP-title
                                         OPTIONAL,
calling-AE-Qualifier                 [7]  AE-qualifier
                                         OPTIONAL,
calling-AP-Invocation-identifier     [8]  AE-invocation-identifier
                                         OPTIONAL,
calling-AE-Invocation-identifier     [9]  AE-invocation-identifier
                                         OPTIONAL,
implementation-information           [29] IMPLICIT Implementation-data
                                         OPTIONAL,
user-information                     [30] IMPLICIT Association-information
                                         OPTIONAL
}
```

(Source: ISO/IEC 8650: 1988)

FIGURE 10.1 ASN.1 Definition of the AARQ APDU

Why are AE invocations and AP invocations identified? Suppose an association is formed between an application entity "Fred" on a computer "Flintstone" and an application entity "Barny" on a computer "Rubble." An integer is assigned to identify the run-time process (the application process, or AP) executing application entity Fred on Flintstone and similarly for application entity Barny on Rubble. Suppose the process containing Fred is unexpectedly terminated, but Barny contin-

ues. When Fred is resurrected as a new run-time process, new invocation identifiers are assigned so that in subsequent communications, Barny will know that it is communicating with a different process than before and hence can determine that the interruption in communication was due to a run-time process failure at Flintstone. Conversely, if communication between Fred and Barny is interrupted, and both processes continue, once communication is restored between Flintstone and Rubble, both application entities will be able to recognize that they are communicating with the same invocation of their peer.

A singularly important parameter passed in the A-ASSOCIATE.request primitive is the *application context name*. The application context name identifies the set of application service elements required by the distributed application initiating this communication.

The information passed to ACSE in the A-ASSOCIATE. request primitive is used to construct a corresponding A-ASSOCIATE request protocol data unit. The ASN.1 definition of this PDU is given in Figure 10.1.

An application that uses the directory system protocol (described in Chapter 7), for example, uses the association control service element and a set of defined remote operations. The application context `directorySystemAC` that describes this is "named" by the OBJECT IDENTIFIER `id-ac-directorySystemAC`; the ASN.1 definition for this application context is:

```
directorySystemAC
APPLICATION-CONTEXT
APPLICATION SERVICE ELEMENTS {aCSE}
BIND DSABind
UNBIND DSAUnbind
REMOTE OPERATIONS {rOSE}
OPERATIONS OF {
chainedReadASE, chainedSearchASE,
chainedModifyASE}
ABSTRACT SYNTAXES {
id-as-acse, id-as-directorySystemAS}
::= {id-ac-directorySystemAC}
```

Thus, when creating an association between two (directory) application processes that wish to use the directory system protocol, the requesting application entity assigns the information object `application-context-name` (one of the data in the AARQ APDU, shown in Figure 10.1) the value of the OBJECT IDENTIFIER `id-ac-directory SystemAC`. Using the application context name, the called application entity determines the set of application service elements that must be available to support this distributed application.

The requester also identifies the abstract syntaxes (modules of

ASN.1 "code"; note that this, too, is derived from the application context), the presentation services that are required, and the presentation address of the called association-control user (a.k.a. the "accepter") in the A-ASSOCIATE.request primitive. These parameters are not conveyed to a remote association control "peer" using the association control protocol. They are submitted in the A-ASSOCIATE.request by the requester so that the association control service provider can make appropriate requests when establishing a presentation connection to support the distributed application; they are used to compose a P-CONNECT.request primitive (see Chapter 11).

Why should the association control service user have knowledge of both the presentation and session service requirements, and why must these be indicated in an A-ASSOCIATE.request? Part of the reason lies in the parallel development of the CCITT X.400 Message Handling System recommendations and the OSI upper layers. CCITT had developed message handling standards with the understanding that the T.62 protocol developed for teletex would be used at the session layer. What Marshall Rose describes in a "soapbox" as "largely hysterical reasons" were simply political: accommodating the perceived (exaggerated) "embedded base" of teletex terminals in X.400 was considered "strategic," and T.62 functions were effectively "grandfathered" into the OSI session layer, rendering moot any argument about correct placement of these functions elsewhere.

What would have been a "correct" solution? There are at least two schools of thought on this subject: (1) the session and presentation functions are in the wrong layers and should have been identified as application layer functions in the first place, perhaps to the extreme of eliminating the session layer entirely; and (2) the functions of the upper layers are not inherently hierarchical, and a complete rethinking of the upper layers was appropriate. In fact, aspects of both schools of thought exist: the pass-through of presentation and session requirements is a concession to the first school of thought (the application layer ultimately controls what goes on in the session layer, and where the protocol operates is a formality) and the definition of application service elements in the OSI Application Layer Structure standard is a concession to the second school of thought (you have building blocks in the application layer).

Association control and presentation connection processing are "piggybacked": the association control APDUs for A-ASSOCIATE, A-RELEASE, and A-U-ABORT are conveyed as user data in corresponding presentation service primitives (P-CONNECT, P-RELEASE, and P-U-ABORT, respectively). The sequence of primitives and application proto-

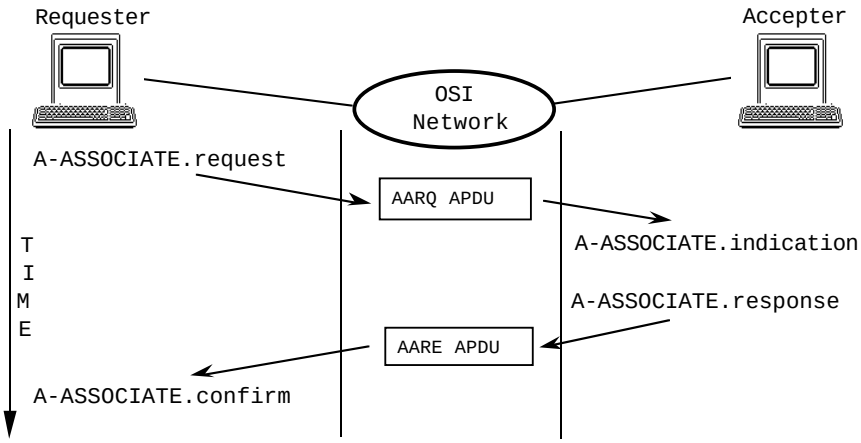

FIGURE 10.2 Association Establishment

col exchanged between a requester and an accepter during association establishment is depicted in Figure 10.2.

A-ASSOCIATE is a confirmed service; thus, in the normal sequence of events, the A-ASSOCIATE.request is processed by the requester, which composes an AARQ APDU and submits it to the presentation layer (via the P-CONNECT.request primitive). The AARQ APDU is delivered (as user data of the P-CONNECT.indication primitive), and the information conveyed in the APDU is passed to the accepter via the A-ASSOCIATE.indication. The accepter may accept or reject the association (admittedly, having an accepter "reject" something seems counterintuitive, but that's what the standard says); this result, together with a reason code, is indicated in the *result* and *diagnostic* parameters[2] of the A-ASSOCIATE.response. The result and diagnostic, together with the identifiers of the responding application entity, are used to compose the A-ASSOCIATE response packet (in Figure 10.2, the AARE APDU). The response packet is delivered to the requester (via the P-CONNECT.confirm primitive). The ASN.1 definition of the AARE APDU is shown in Figure 10.3.

2. According to the standard, the reason for rejection may be "permanent" or "transient," but what constitutes permanent or transient is not discussed. The choice of diagnostic codes defined in the ACSE protocol includes "application context name not supported" and a set of codes indicating "title/invocation/qualifier not recognized." The OSI folks did try to standardize the use of these and additional reason codes but failed. Transient is typically used for no-resource conditions, and permanent is used for cases in which the named AP/AE doesn't exist or requested session user requirements, for example, aren't supported.

```
AARE-apdu ::=                [APPLICATION 1] IMPLICIT SEQUENCE
{ protocol-version                           [0] IMPLICIT BIT STRING
                                                 { version1 (0) }
                                                 DEFAULT version1,
application-context-name                     [1] Application-context-name,

result                                       [2]    Associate-result,

result-source-diagnostic                     [3]    Associate-source-diagnostic,

responding-AP-Title                          [4]    AP-title
                                                    OPTIONAL,
responding-AE-Qualifier                      [5]    AE-qualifier
                                                    OPTIONAL,
responding-AP-Invocation-identifier          [6]    AP-invocation-identifier
                                                    OPTIONAL,
responding-AE-Invocation-identifier          [7]    AE-invocation-identifier
                                                    OPTIONAL,
implementation-information                   [29]   IMPLICIT Implementation-data
                                                    OPTIONAL,
user-information                             [30]   IMPLICIT Association-information
                                                    OPTIONAL
}
```

(Source: ISO/IEC 8650: 1988)

FIGURE 10.3 Abstract Syntax of AARE-apdu ASN.1 Definition of the AARE APDU

No "data packets" are defined for the association control service element. Once an association is established, no association control protocol is exchanged until the association is released or aborted; only the protocol of other application service elements, including the user element, is exchanged by the application processes.

The *A-RELEASE* service (Figure 10.4) is used to terminate an association in an orderly manner; the communicating application entities typically see that all information exchanges have been completed prior to releasing the association. This is a confirmed service. It uses the A-RELEASE request and A-RELEASE response packets (in Figure 10.4, the RLRQ and RLRE APDU*s*, respectively), and the service primitive and protocol action is similar to that of the A-ASSOCIATE service.

The *A-U-ABORT* service (Figure 10.5) is disruptive; it is invoked when one association control user chooses to terminate an association abruptly. No attempt is made to assure that all information exchanges have been completed. *A-P-ABORT* occurs when the association control service provider releases, either due to errors internal to the association control service provider or when the association control provider determines that services below the application layer have failed in some man-

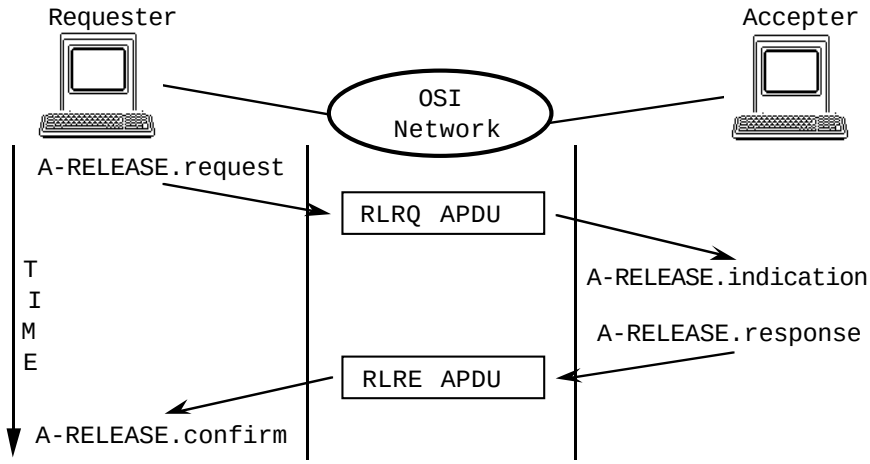

FIGURE 10.4 Association Release

ner. Both ABORT services use the A-ABORT packet (in Figure 10.5, the ABRT APDU) when protocol is exchanged; A-P-ABORT does not involve protocol exchange when occurring as the result of communication failure.

Modes

ISO/IEC 8649 defines two "modes" of ACSE operation: in the *normal mode,* implementations use the association control service and protocol as defined in ISO/IEC 8649 and 8650; in the *X.410-1984 mode,* however, implementations are required to pretend that the association control protocol doesn't exist for the sake of communicating with CCITT X.410-1984 implementations. In such circumstances, implementations pass the presentation and session service requirements through to the presentation

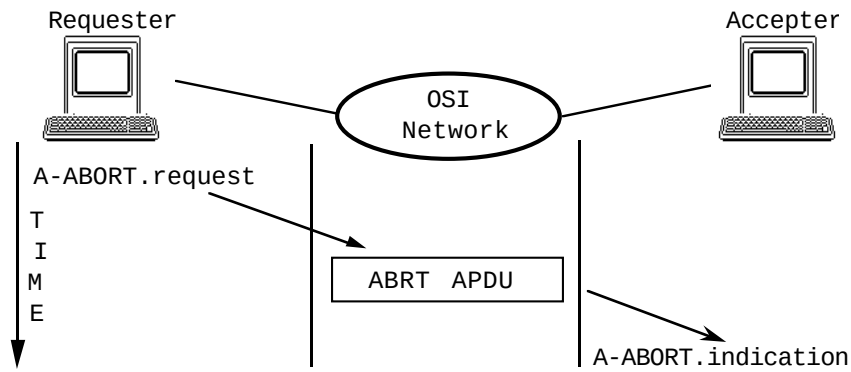

FIGURE 10.5 Association Abort (User-initiated)

layer and use only the presentation kernel (see Chapter 11). No association control protocol is exchanged; the mode of operation is conveyed during the presentation connection establishment (see Chapter 11).

Only a handful of the parameters of the A-ASSOCIATE service are used in the case of X.410-1984 mode (those that are not marked with "*" in Table 10.1). The conformance statement for the association control protocol says that either or both modes must be implemented.

The X.410-1984 mode of operation is named after the 1984 version of the CCITT X.400 Message Handling System recommendations. The history behind the existence of two modes of ACSE operation is simple. In 1983, the CCITT was in the process of completing its recommendations for message handling services; at the time, however, the disposition of function between the application and presentation layers of the message handling architecture was still a subject of great debate in the ISO OSI community. Since leap years were historically "publish or perish" times for CCITT—the member organizations had to agree upon recommendations during a plenary session or wait another four years for the next plenary—it was determined that the best interests of the CCITT community would not be served by waiting for the OSI upper layer organization to be completed. Applying the changes throughout the MHS architecture could be done in 1988, with only the small issue of "backward compatibility" to consider—and how hard could that be?

Reliable Transfer Service Element

There are very few occasions when you have the luxury of reading a book uninterrupted, or "cover to cover." More often than not, you find time to read several pages, perhaps a chapter or two, but inevitably, your reading time will be interrupted, or you'll fall asleep. You might dog-ear the last page read or use a bookmark to locate the point where you should continue when you once again find time to read. You do so because you would certainly prefer to continue reading from the page where you left off rather than begin again from page 1. In such situations, you and the book have formed an association, and your use of a bookmark, for example, provides a *synchronization* point (or "checkpoint"). Similarly, if you engage in a telephone conversation with someone and your call is unexpectedly disconnected, you redial and typically continue, or *recover,* the conversation from the point at which you were interrupted. Again, you and the party you call are engaged in a (voice)

application, and your ability to remember the last subject under discussion serves as a synchronization point in the conversation in the event that the underlying telephone connection fails. In both examples, large amounts of information are exchanged between communicating entities. And although the two forms of dialogue illustrated differ—the first is monologue, the latter two-way alternate—there is a strong incentive in both cases to avoid repetition.

End users of applications that exchange large amounts of data—multimedia message handling, image transfer/retrieval, office or electronic document interchange, plain old bulk file transfer—share this incentive; they don't want to "start from scratch" if network connections fail. It wastes time and resources and is potentially costly. The *reliable transfer service element* (RTSE) (ISO/IEC 9066-1: 1989; ISO/IEC 9066-2: 1989) provides services that enable application entities to (1) synchronize and later recover from (temporary) communications failures between end systems with a minimum amount of retransmission, and (2) conduct both monologue and two-way alternate conversations without having to deal directly with the mechanics of providing such services.

The reliable transfer service element relieves the user from having to worry about the details of association establishment and release. This service does not provide low-level control of associations the way that association control does; rather, it initiates and terminates an association for a reliable transfer service user with the primary purpose of transferring data. The reliable transfer service effectively bundles several layers of service features into a single service. Using the RTSE primitives, a reliable transfer service user can:

- Coordinate information exchange within an association (turn management)
- Confirm that a remote reliable transfer service user has received and secured data before sending more data (synchronization)
- Recover from the temporary loss of a presentation connection
- Ensure that an association is closed without loss of any information that might have been in transit when the close was requested (the popular term for this mechanism is *graceful* close)

The foregoing can be accomplished with three simple service requests: RT-OPEN, RT-TRANSFER, and RT-CLOSE.

As an example, consider a medical-imaging service, one that enables physicians to retrieve X rays or magnetic resonance images (MRIs) from a central database across a network providing application-to-application throughput of 1 Mbps. MRIs are very large, measurable in mega-

bytes of data. In a worst-case scenario—all but the last 1,000-byte fragment of the image is transferred, and the network connection fails — retransmission of a 10-megabyte (80-megabits) MRI "from scratch" could cause nearly 80 additional seconds of delay in the transfer. Now, a physician's time is precious, and such delays might not be tolerated. The reliable transfer service could be used by this medical-imaging service element to assure that the precise image is delivered to the physician without excessive retransmission. In this example, retransmission could begin from the point of failure (the last 1,000 bytes), and the delay would be greatly reduced.

The reliable transfer process, from request to completion of transfer, is composed of the functions discussed in the following sections.

Association Establishment

A physician makes a request for an MRI from a remote image database. The user element of the remote image database application invokes the RT-OPEN service, which uses the Association-Control Service Element to form an association between the computer hosting the database and the physician's local computer (see Figure 10.6). Note that in this example, the information object `.application-context-name` of the AARQ APDU would be assigned the OID value for the association context name of the mythical MRI application "davesExcellentMRIs." Since the origin and location of this name information are not obvious, let's consider the example a bit further. Suppose "Dave's Excellent Software" is a U.S.-based company, and suppose further that the firm decides to obtain a globally unique identifier from the U.S. registration authority (see Chapter 5). The organization identifier is given the alphanumeric representation of "davesExcellent" and assigned a numeric value of "22345." When Dave's Excellent Software develops the MRI application, it uses the branch of the object identification tree `iso (1) member-body (2) US (840) davesExcellent (22345)` to uniquely identify application entities, abstract syntaxes, application context names, etc., for its applications. Thus, depending on how the company decided to write ASN.1 definitions for its applications, the application context name of the application davesExcellentMRIs might look like this:

```
davesExcellent-MRIs-ac OBJECT IDENTIFIER
::= { iso (1) member-body (2) US (840) davesExcellent (22345)
     MRIs (01) applicationContext (03) }
```

This application context name identifies the user element[3] and, by impli-

3. The term *user element* refers to the consumer of ASE services. (ISO/IEC 7498-1: 1993).

cation, the set of application service elements that the user element requires[4] (in this case, association control service element and RTSE).

TABLE 10.2 RT-OPEN Service Parameters

Parameter Name	*Request*	*Indication*	*Response*	*Confirm*
Dialogue mode	Mandatory	Mandatory(=)		
Initial turn	Mandatory	Mandatory(=)		
Application protocol	User option	Conditional(=)		
User data	User option	Conditional(=)	User option	Conditional(=)
Mode	A	A		
Application context name	A	A	A	A
Calling AP title	A	A		
Calling AE qualifier	A	A		
Calling AP invocation identifier	A	A		
Calling AE invocation identifier	A	A		
Called AP title	A	A		
Called AE qualifier	A	A		
Called AP invocation identifier	A	A		
Called AE invocation identifier	A	A		
Responding AP title			A	A
Responding AE qualifier			A	A
Responding AP invocation identifier			A	A
Responding AE invocation identifier			A	A
Result			A	A
Result source				A
Diagnostic			A	A
Calling presentation address	P	P		
Called presentation address	P	P		
Responding presentation address			P	P
Presentation context definition list	P	P		
P-context definition result list			P	P
Default presentation context name	P	P		
Default presentation context result			P	P

Note: In this table, the notation "A" is used to identify parameters that appear in the RT-OPEN primitive so that their values can be "passed through" RTSE to corresponding ACSE primitives, and the notation "P" is used to identify parameters that are similarly "passed through" to the presentation service.

4. Note that the term *application protocol data unit* is used generically here. Depending on how the application is written, the APDU may be an ASN.1 SEQUENCE consisting of

The abstract syntaxes that the presentation layer would have to manage would include ACSE, RTSE, and the APDUs defined for the MRI transfer. Thus, the abstract syntax component of the *presentation context definition list* parameter of the A-ASSOCIATE.request would contain the following object identifier value:

```
davesExcellentMRIs-as OBJECT IDENTIFIER
::= { iso (1) member-body (2) US (890) davesExcellent
    (22345) MRIs (01) abstract syntax (02) }
```

It would also contain the object identifier values for the ACSE and RTSE abstract syntaxes.

The protocol necessary to establish an association via the reliable transfer service—the RT-OPEN request and accept packets (in Figure 10.6, the *RTORQ* and *RTOAC APDUs*, respectively)—is conveyed as user data in the A-ASSOCIATE service primitives and "piggybacked" onto the *AARQ* and *AARE APDUs*. In effect, the reliable transfer protocol "header" merely adds to the list of parameters negotiated during association control; it does not create an additional association. The state machine of the RTSE protocol is wedded to that of the association control protocol during association setup and release. The additional parameters include *checkpoint size*, the number of kilobytes of data that may be sent between synchronization points, and *window size*, an upper bound on the amount of data that can be sent. Checkpoint size is an indication of how frequently you want to insert synchronization points in the data stream; window size indicates how many units of checkpoint size you'll send before you will wait to hear that the receiver has secured the data you have transmitted—in effect, how far you are willing to press your luck.

Once an association is established, the image is transferred using the RT-TRANSFER service.

RT-TRANSFER: Activity Management, Checkpointing, and Synchronization

A reliable transfer service user can submit arbitrarily large amounts of data—as a single, encoded user element APDU value—in a single RT-TRANSFER.request. In our medical image transfer example, an entire image of perhaps several megabytes of data might constitute a single user element APDU. The RT-TRANSFER service accepts such arbitrarily large quantities of user data and treats each submission as a separate transfer *activity*. Within the context of this transfer activity, a sending

an operator (e.g., FETCH-IMAGE), followed by the arguments patient name (OCTET-STRING) and patient number (INTEGER). An associated medical image (GRAPHICS STRING or BITSTRING) might be retrieved by using the patient information to identify which image "instance" is requested.

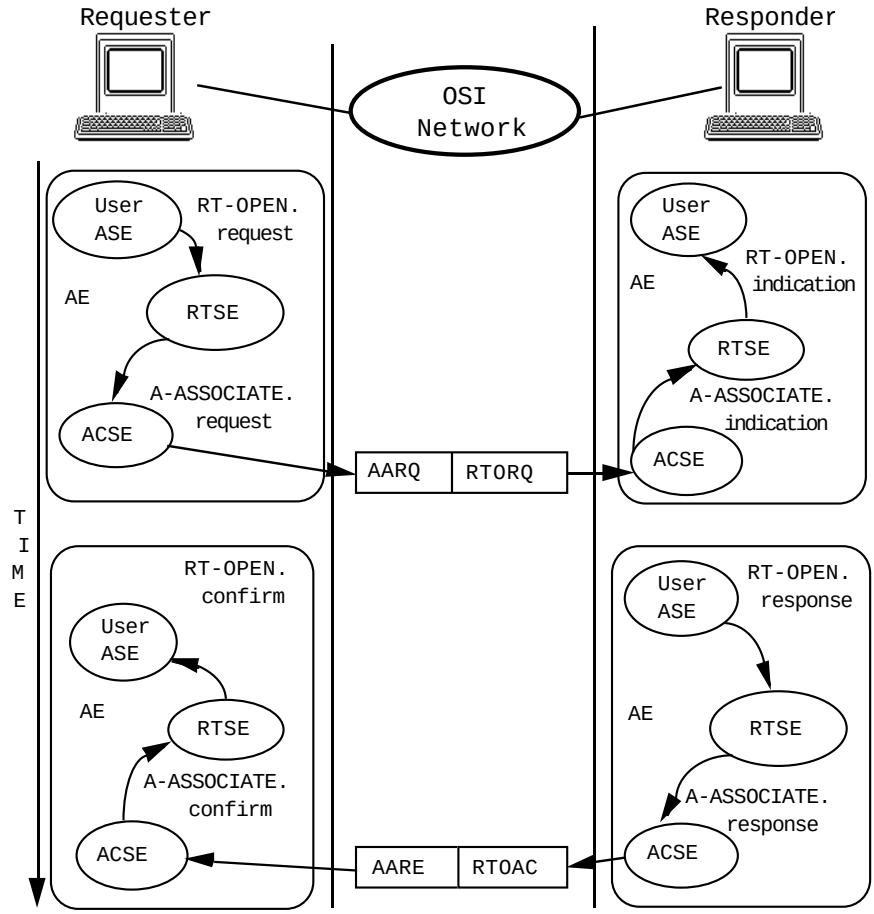

FIGURE 10.6 Reliable Transfer Open

RTSE fragments the medical image into octet strings of length {checkpoint size × 1,024} and then invokes the following presentation services (see Chapter 11):

- P-ACTIVITY-START.request to signal the beginning of a user element APDU (an activity) to the receiver
- P-DATA.request to transmit the first fragment as a RELIABLE TRANSFER APDU (in Figure 10.7, the initial RTTR APDU)

The sending RTSE then alternately sends:

- A P-MINOR-SYNCHRONIZE.request to insert a checkpoint into the information stream

- A P-DATA.request to transmit the next (final) fragment as a RELIABLE TRANSFER APDU (in Figure 10.7, the subsequent RTTR APDUs)

until all the user data (the entire medical image) have been transferred. Finally, the sending RTSE uses the P-ACTIVITY-END.request primitive to mark the end of the activity when the last fragment of the medical image has been sent.

On the receive side, the RTTR APDUs arrive as user data transferred across a presentation connection. The receiving reliable transfer service element secures data extracted from each RTTR PDU received and confirms this by responding to the P-MINOR-SYNCHRONIZE.indications. The receiving RTSE user (the user element of the physician's medical-imaging application process) is informed when transfer of the

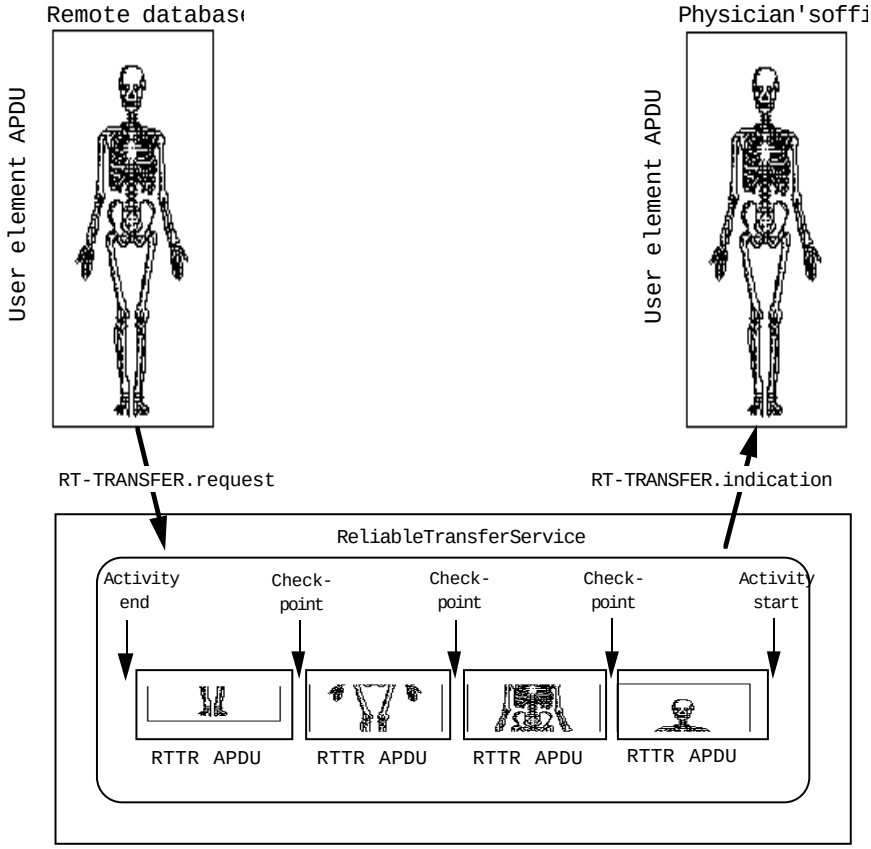

FIGURE 10.7 Reliable Transfer

entire image is complete (the end of this transfer activity) via the RT-TRANSFER.indication primitive. If the transfer activity is completed within a *transfer time* indicated by the sender (a "catastrophic failure" time-out value), the sending RTSE user (the remote image database application) is informed that the user data have been secured by the receiving RTSE via the RT-TRANSFER.confirmation primitive. If the transfer is not completed within the specified transfer time, the sending RTSE gives up, invokes the presentation activity discard service (see Chapter 11) to clean up the mess, and aborts the association.

Turn (Token) Management

The medical image transfer example considers only the case in which reliable transfer is a one-way operation (from remote database to physician). Other applications might want to organize information exchange into dialogues (two-way alternate). In such communications, elements familiar to telephone conversations—politely waiting one's turn to speak, asking for a turn to speak, interrupting, resuming after an interruption—are implemented using RT-TRANSFER services in conjunction with RT-TURN-PLEASE and RT-TURN-GIVE services; these services make use of the turn-management presentation (and session) services discussed in Chapter 11.

Reliability Mechanisms: Exception Reporting/Transfer Retry, Association Recovery/Transfer Resumption

The reliable transfer service attempts to mask several types of error conditions from reliable transfer service users during a transfer activity. For example, the reliable transfer service performs recovery of a single activity without user intervention: if a receiving RTSE encounters an error that is procedural or local to the receiving RTSE and is recoverable, it will discard the present activity and retry the transfer using exception and activity-management presentation services. The reliable transfer service also recovers from the temporary loss of an association between two reliable transfer service users by resurrecting the association (here's where invocation identifiers are handy) and resuming an activity interrupted by the communications failure from the last checkpoint. Of course, when the error detected by a receiving RTSE is "severe," or when an association cannot be resurrected within a user-specified *recovery time*, the service will be aborted. The RT-ABORT service is also available to allow abrupt termination of an association between two reliable transfer service users (in Figure 10.8, the ABORT packet for reliable transfer, the RTAB APDU, is piggybacked onto the association control ABORT APDU, ABRT).

RT-CLOSE: Association Release

The RT-CLOSE service provides a "graceful" termination of an association. Any outstanding RT-TRANSFER.request must be confirmed (in

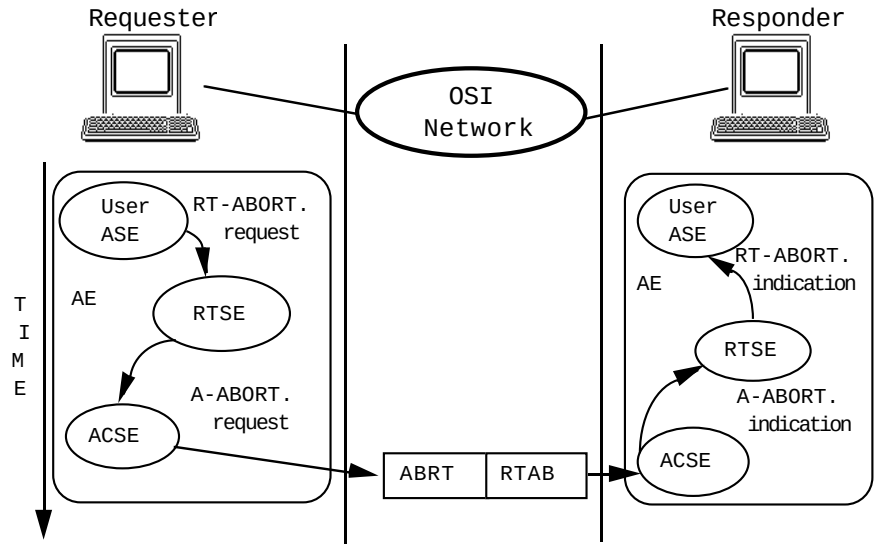

FIGURE 10.8 RT-ABORT (User-initiated)

both directions) before the association can be closed; i.e., any activities in the process of being transferred must be completed. Figure 10.9 illustrates how the RT-CLOSE service maps onto the A-RELEASE service. Note that no explicit RTSE protocol is used in the close process (recall that the RLRQ APDU is part of the association control protocol); user data from the user element are passed as user data in the A-RELEASE primitives and protocol. Although this might appear to be a curious departure from the user-initiated RT-ABORT, in which an explicit RTAB APDU may be used, it's really OK; the RTAB APDU allows user elements to convey an APDU that might provide a more detailed explanation of the reason for abort than the Abort-reason "user-error" specified in ISO/IEC 9066-2.

One might expect that OSI file transfer, access, and management would make excellent use of the reliable transfer service. Unfortunately, FTAM was developed before the reliable transfer service, and FTAM manages presentation services directly. There are some who consider this "a good thing," since FTAM makes extensive use of presentation services beyond those used by the RTSE.

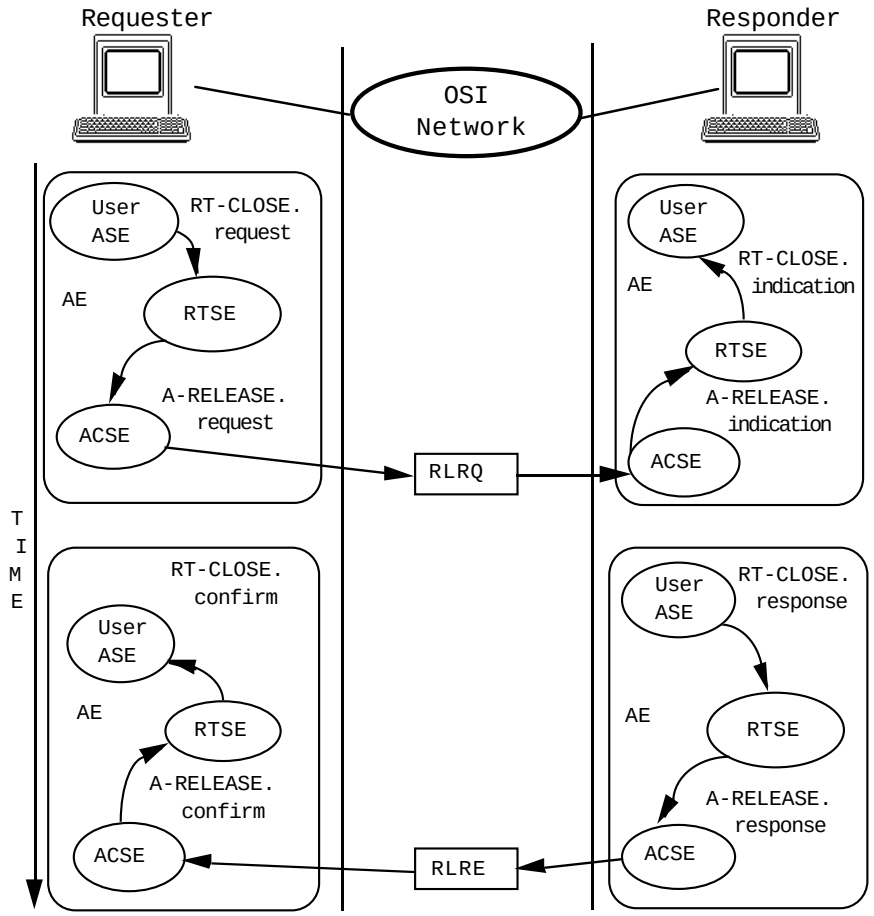

FIGURE 10.9 Reliable Transfer Close

Remote Operations Service Element

As the name implies, the remote operations service element (ROSE) allows an application process on one computer to invoke an application process on a different (remote) computer to perform some set of operations. The concept is a powerful one and is used in applications where client/server, manager/subordinate (agent), or multipeer, shared processing relationships are necessary.

Remote operations, often called *remote procedure calls*, appear in many distributed-processing applications today. The Network File System (NFS; Sandberg 1988) applies a client/server remote operation paradigm in providing a networked or distributed UNIX file system. Client

workstations (often diskless) invoke the services of a "disk-full" computer, called a file server, to read information from, and write information to, long-term storage. Network management applications based on OSI common management use remote operations to invoke the services of subordinate or agent applications that run on manageable network resources to isolate and correct faults, configure those resources, and collect management information through some set of management operations. And both the OSI Message Handling System and the OSI Directory (see Chapters 7 and 8) make extensive use of remote operations in the somewhat similar user agent/transfer agent relationships.

Some distributed operating system applications—e.g., DUNE (Alberi and Pucci 1987), LOCUS (Walker et al. 1983)—apply the client/server paradigm to a group of networked computers, whereas other distributed operating systems behave as if they were a single "monster" multiprocessor, multitasking (super)computer, with each computer possessing the ability to pass off a task or procedure to another, less busy computer and each having "equal access" to the others' resources (CPU, disk, etc.).

Just as the reliable transfer service element simplifies the process of reliable data transfer between computer systems, the remote operations service element simplifies the process of distributing operations across multiple computer systems. Like RTSE, ROSE relieves user elements from having to worry about the details of association establishment and release (it either invokes association control itself or allows the reliable transfer service element to do so) and bundles several layers of service features into a single service. Within the context of an association, ROSE allows an application process on a local computer to invoke an application process on a remote computer and request that it perform an operation. The operations themselves are treated as ASE services and are defined as abstract syntaxes in standard application service elements like the MHS and Directory ASEs or in user-programmed application service elements. (Remote operations in OSI thus operate on abstract rather than concrete data types.) OSI collectively labels the set of ASE services available to a user element of an application entity an *operations interface* (Figure 10.10).

In the figure, each "user ASE" represents some remote operation(s) the user element may invoke. The user element may use the ACSE directly to establish associations between itself and other user elements. The user element may optionally use the reliable transfer service element to handle association control and reliable transfer if information transfer between user elements would benefit from this additional facility.

Synchronous and Asynchronous Operations

In some distributed applications, the invoking user element may wish to wait for the operation to be completed (remote operations performed in this "lockstep" fashion are classified as *synchronous operations*), or the user element may invoke a remote operation and continue processing while the operation is performed (*asynchronous operation*). In both cases, the performer can be directed to return indications of whether or not the requested operation was successful (a result and/or an error). The matrix of synchronous/asynchronous modes and reply possibilities is illustrated in Table 10.3; each combination is known as an *operation class*.

TABLE 10.3 ROSE: Operation Classes

Operation Class	*Synchronous*	*Asynchronous*	*Report Result*	*Report Error*
1	✓		✓	✓
2		✓	✓	✓
3		✓		✓
4		✓	✓	
5		✓		

Linked Operations In some distributed applications, the performer of a remote operation may find it necessary to invoke a related operation from the user element that originated the remote operations relationship (the "parent" of the remote operation). Remote operations *linked* to the parent remote operations are called "child" operations. Figure 10.11 illustrates this concept.

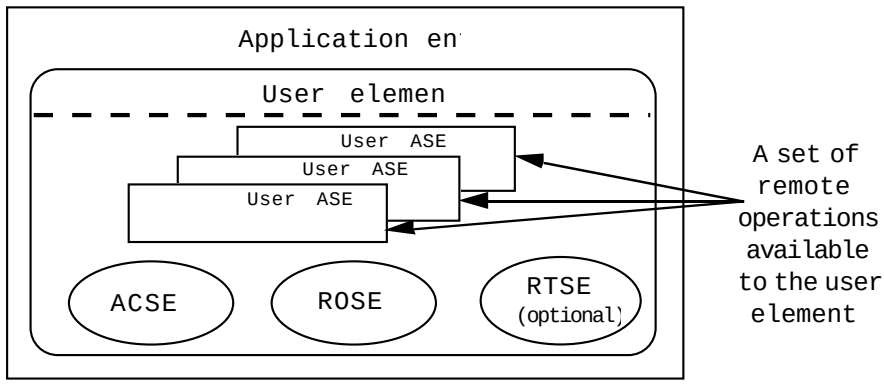

FIGURE 10.10 Operations Interface

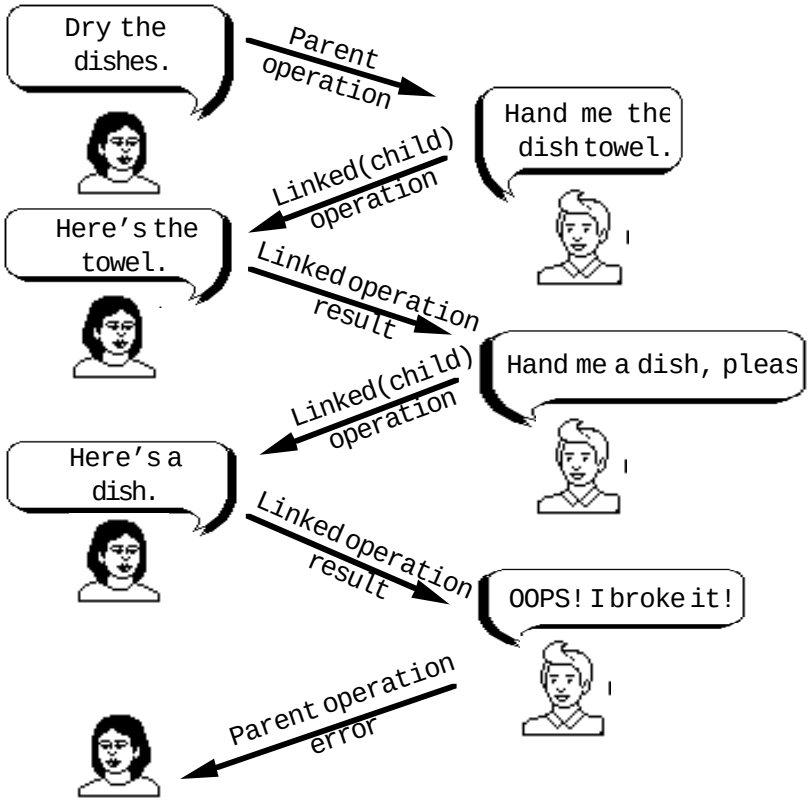

FIGURE 10.11 Linked Operations

Distributed applications correlate linked operations using the *invoke ID*. A concrete example: remote operations invoke IDs are used to correlate OSI common management information service requests and responses—in other words, management applications assign unique numbers to their requests and use these numbers to decipher later responses from managed systems. For example, when a management application issues an M-GET.request with invoke ID "42," the manager sends an ROIV-M-GET request packet. The agent on the managed system responds with an RORS-M-GET response PDU, also containing the invoke ID "42" (these common management information services call the RO-INVOKE and RO-RESULT ROSE services, which are described later in this chapter). If multiple replies are needed, the agent responds with several ROIV linked-reply packets (each containing the linked ID "42"), concluded by the final RORS-M-GET response packet.

Relationship of ROSE to Other "Core" Application Service Elements

ROSE may use ACSE directly to establish associations. In such configurations, the ROSE user invokes association control to establish and release associations, as illustrated in Figures 10.2 and 10.4, respectively, and uses the presentation layer to transfer data. ROSE users may wish to take advantage of the services of RTSE; in this case, the ROSE user establishes and releases associations by invoking the services of RTSE (RT-OPEN and RT-CLOSE, illustrated in Figures 10.6 and 10.9, respectively) and uses the RT-TRANSFER service (Figure 10.7) to transfer data (see Figure 10.12).

ROSE Services and Protocol

The ROSE services provide the basic elements for interactive (inquiry/response) communications:

- *RO-INVOKE* allows an AE to request that another AE perform an

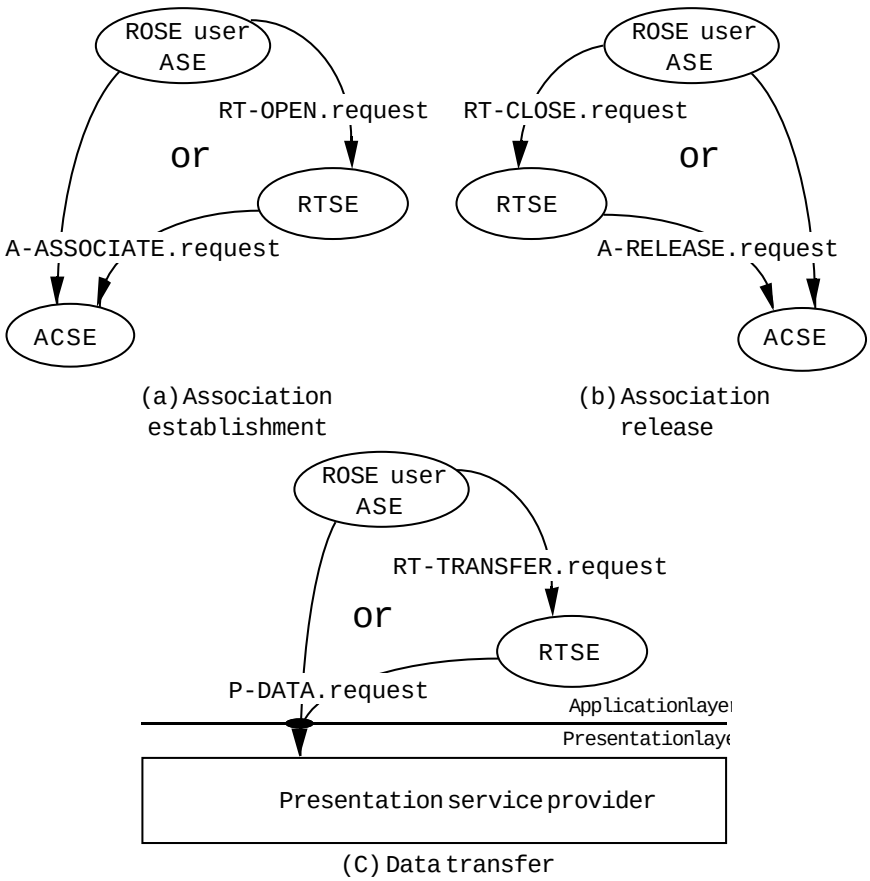

FIGURE 10.12 ROSE Use of "Core" Application Service Elements

operation; the requesting AE is called the *invoker*, and the AE responsible for the grunt work is called the *performer.*

- *RO-RESULT* and *RO-ERROR* allow performers to indicate whether the operation succeeded or failed.
- *RO-REJECT-U* and *RO-REJECT-P* are exception services initiated by the performing AE or ROSE, respectively.

All ROSE services are described in the standards as "unconfirmed" (except RO-REJECT-P, which is provider-initiated). (Table 10.4 illustrates the primitives and parameters of the ROSE.) A more accurate description is that RO-RESULT, RO-ERROR, and RO-REJECT are "confirmations" for operations that are RO-INVOKEd. A typical sequence of ROSE events, for example, is "Send an RO-INVOKE and get back an RO-RESULT or RO-ERROR." RO-RESULT means complete and successful, whereas RO-ERROR means something went wrong—either partial or total failure occurred (for example, the operation may have been performed with a diagnostic returned).

TABLE 10.4 ROSE Primitives

RO SERVICE	*Parameter Value*	*Request*	*Indication*
RO-INVOKE	Operation value	Mandatory	Mandatory(=)
	Operation class	User option	
	Argument	User option	Conditional(=)
	Invoke ID	Mandatory	Mandatory(=)
	Linked ID	User option	Conditional(=)
	Priority	User option	
RO-RESULT	Operation value	User option	Conditional(=)
	Result	User option	Conditional(=)
	Invoke ID	Mandatory	Mandatory(=)
	Priority	User option	
RO-ERROR	Error value	Mandatory	Mandatory(=)
	Error parameter	User option	Conditional(=)
	Invoke ID	Mandatory	Mandatory(=)
	Priority	User option	
RO-REJECT-U	Reject reason	Mandatory	Mandatory(=)
	Invoke ID	Mandatory	Mandatory(=)
	Priority	User option	
RO-REJECT-P	Invoke ID	ROSE provider option	
	Returned parameters	ROSE provider option	
	Reject reason	ROSE provider option	

The ROSE user APDUs are the basis for *the actual work to be done*. The ROSE protocol is merely a "wrapper," present primarily to convey parameters of the service. Specifically, the remote operations invoke packet (ROIV APDU) is used to convey the *invocation identifier, linked invocation identifier*, and *operation value* from the RO-INVOKE.request. The ASN.1 data type identified in the ARGUMENT clause of the remote operation may be included in the ROIV APDU as well. Other parameters (the *operation class* and the relative *priority* of this operation with respect to other invoked operations) are not carried in protocol but affect handling by the local ROSE provider.

The remote operations *result* (RORS) and remote operations error (ROER) packets convey the invocation identifier as a means of correlating the ASN.1 data type identified in the RESULT and ERROR clauses of an OPERATION to a given invocation of that remote operation.

The remote operations reject (RORJ) packet again conveys the invocation identifier as a means of correlating a user or provider rejection of an operation. A reason for rejecting the operation is encoded as an ASN.1 data type called *problem* (for which ISO/IEC 9072-2 provides a list of standard values).

RO-Notation

ROSE provides a set of ASN.1-defined macros—BIND, OPERATION, UNBIND—to facilitate association establishment, RO invocation, and association release. Two additional macros—APPLICATION-CONTEXT and APPLICATION-SERVICE-ELEMENT—assist programmers in defining the application context for a distributed application and in defining ROSE user elements, respectively.

Let's assume that the object identifier for the whimsical user element "dish drying" from Figure 10.11 is as follows:

```
dishDryingService OBJECT IDENTIFIER
::= { iso bogus (999) example (999) dishDrying (99) }
```

The APPLICATION-CONTEXT macro might then look like this:

```
dishDryingContext                   APPLICATION-CONTEXT
    APPLICATION SERVICE ELEMENTS    { aCSE }
    BIND                            dishDryerEnslave
    UNBIND                          dishDryerFreeToCruise
    REMOTE OPERATIONS               { rOSE }
    INITIATOR CONSUMER OF           { dishDryingASE }
    ABSTRACT SYNTAXES               { aCSE-abstract-syntax,
                                    dryTheDishes-abstract-
                                    syntax }
    ::= { iso bogus (999) example (999) dishDrying (99)
         application-context (2) }
```

```
-- the OID value for the abstract syntax --
dryTheDishes-abstract-syntax OBJECT IDENTIFIER
    ::= { iso bogus (999) example (999) dishDryer (99)
         abstract-syntax (1) }
```

In this example, the application context identifies association control, the BIND and UNBIND macros, the ROSE (as the remote operations provider), and the ROSE user application service element "dish-Drying ASE" (the definitions of the actual remote operations to be performed by the initiator and responder). In this example, the INITIATOR CONSUMER OF clause indicates that the ROSE user that establishes the association may invoke "dishDryingASE". The clauses RESPONDER CONSUMER OF and OPERATIONS OF (not used in this example) could be used to indicate ROSE user application service elements that only the responder, or both the initiator and the responder (respectively) may invoke.

The object identifier value for the application context is passed as a parameter in association establishment (as the application context name parameter of the A-ASSOCIATE.request primitive). Similarly, the abstract syntaxes of all the application service elements—representing the set of APDUs required for this application—are passed as a parameter in association establishment (the OIDs of `aCSE-abstract-syntax` and `dryTheDishes-abstract-syntax` are enumerated in the presentation context definition list parameter of the A-ASSOCIATE.request primitive).

The "dishDryingASE" is defined using the APPLICATION-SERVICE-ELEMENT macro:

```
dishDryingASE           APPLICATION-SERVICE-ELEMENT
    CONSUMER INVOKES     { dryTheDish }
    ::= { iso bogus (999) example (999) dishDrying (99)
        application-service-element (3) }
```

Here, the clause CONSUMER INVOKES identifies the parent remote operation "dryTheDish"; this is the only remote operation that Mom can ask the SUPPLIER (son Billy) to perform. If there were parent operations that son Billy could invoke, they would be identified using the SUPPLIER INVOKES clause, and if there were parent operations that both the CONSUMER and the SUPPLIER could invoke, they would be specified using the OPERATIONS clause.

The BIND and UNBIND macros for this example might look like this:

```
dishDryerEnslave
    BIND
            ARGUMENT ::= bindArgument
            RESULT ::= bindResult
            BIND-ERROR ::= bindError
```

```
dishDryerFreeToCruise
    UNBIND
            ARGUMENT ::= unbindArgument
            RESULT ::= unbindResult
            UNBIND-ERROR ::= unbindError
```

In this example, both the BIND and UNBIND macros are defined as synchronous operations: Mom asks son Billy to dry the dishes, expecting compliance or an argument, and also expecting an indication that the task is completed before she will allow Billy to "cruise." There are, of course, error situations (the actual definition of the parameters for these remote operation macros is left to the readers' imagination).

The BIND macro hides the details of establishing an association from the ROSE user, irrespective of whether the association control service element is used directly (Figure 10.2) or the association is established via the reliable transfer service (Figure 10.6). All the information necessary to establish an association is identified in the APPLICATION-CONTEXT and APPLICATION-SERVICE-ELEMENT macros noted earlier. The UNBIND macro is used to release the association (once the dishes are dried).

Linked Operations

One expects to find several linked operations associated with the parent operation "dryTheDishes" identified in the CONSUMER INVOKES statement of the application service element definition:

```
dryTheDishes ::= OPERATION
    BIND
        ARGUMENT ::=
        RESULT ::=
        ERROR ::=
        LINKED ::= { askForTowel, askForDish, howManyLeft }
        ::= 0

askForTowel  AskForTowel  ::= 1
askForDish   AskForDish   ::= 2
howManyLeft  HowManyLeft  ::= 3
```

The linked operations identify work that the CONSUMER (Mom) may be called upon to do by the SUPPLIER (Billy). Note that each parent OPERATION may have local or globally unique values; i.e., the data type of operations may be INTEGER (as illustrated) or OBJECT IDENTIFIER. The linked operations are identified within the context of the parent operation.

The ASN.1 specification of each OPERATION formally defines the work a performer must attempt to complete, in a machine-independent

fashion. There are two ways that ROSE might be used to invoke these operations:

- The consumer can pass the identification of a remote operation (and associated results/errors) to a supplier, which effectively points the supplier to a concrete set of procedures to execute. The input to these concrete procedures and anticipated replies (results/errors) accompany this identification (the ARGUMENT, RESULT, and ERROR clauses).
- The consumer can pass the identification of a remote operation and accompany this with an ASN.1-encoded copy of the machine-specific software that is to be executed. For example, one computer in a UNIX-based distributed operating system might use ROSE to schlepp a user C program fragment off to a less busy peer as the ARGUMENT data type in the OPERATION.

Putting It All Together

Relating all the pieces in a normal sequence of events:

1. A user element invokes a remote operation via the BIND macro.
2. If reliable transfer service is identified in the APPLICATION-CONTEXT macro, ROSE establishes an association between an initiator and a responder using the RT-OPEN service. If reliable transfer service is not identified in the APPLICATION-CONTEXT macro, ROSE establishes the association using the A-ASSOCIATE service. (Note that a BIND could be disrupted by any underlying ABORT service.)
3. If the BIND is successful, the initiating or responding user element may invoke remote operations as described in the {CONSUMER INVOKES, SUPPLIER INVOKES, OPERATIONS} clauses of the APPLICATION-SERVICE-ELEMENT macro using the OPERATION macro. ROSE processes the OPERATION macro using the RO-INVOKE, RO-RESULT, and RO-ERROR services. If RTSE is used, the RO-INVOKE service is mapped onto the RT-TRANSFER service; otherwise, the ROSE makes direct use of presentation data transfer.
4. The invocation of any of these parent operations may result in the invocation of linked operations. Child operations may appear as multiple replies, resulting in a sequence, for example, that begins with sending an RO-INVOKE, continues with the receipt of several RO-INVOKEs, and is concluded by one final RO-RESULT or RO-ERROR.
5. An OPERATION may be disrupted via the RO-REJECT service if

ROSE or a ROSE user cannot process the OPERATION. (An OPERATION could also be disrupted by any underlying ABORT service.)

6. When the user elements have exhausted their use of ROSE, the UNBIND macro is used to release the association. If the reliable transfer service is used, ROSE invokes the RT-CLOSE service; otherwise, ROSE invokes the A-RELEASE service. (Again, association release can be disrupted by any underlying ABORT service.)

ROSE is arguably the most powerful and frequently used application service element (ACSE is frequently used but not nearly as powerful). The whimsical dish-drying example illustrates how one goes about constructing remote operations. Figure 10.13 presents the ASN.1 definition of the directory system protocol, which was ROSE as an example.

```
DirectorySystemProtocol {joint-iso-ccitt ds(5) modules(1) dsp(12)}
DEFINITIONS ::=
BEGIN

EXPORTS
     directorySystemAC, chainedReadASE, chainedSearchASE,
     chainedModifyASE;

IMPORTS
     distributedOperations, directoryAbstractService
          FROM UsefulDefinitions {joint-iso-ccitt ds(5) modules(1)
               usefulDefinitions(0)}
APPLICATON-SERVICE-ELEMENT, APPLICATION-CONTEXT, aCSE
               FROM Remote-Operations-Notation-extension {joint-
iso-ccitt
                    remote Operations(4) notation-extension(2)}
     id-ac-directorySystemAC, id-ase-chainedReadASE,
     id-ase-chainedSearchASE,
     id-ase-chainedModifyASE, id-as-directorySystemAS, id-as-acse;
          FROM ProtocolObjectIdentifiers {joint-iso-ccitt ds(5)
modules(1)
               protocolObjectIdentifier(4)}
     Abandoned, AbandonFailed, AttributeError,
     NameError, SecurityError, ServiceError, UpdateError
          FROM DirectoryAbstractService directoryAbstractService

     DSABind, DSAUnbind,
     ChainedRead, ChainedCompare, ChainedAbandon,
     ChainedList, ChainedSearch,
     ChainedAddEntry, ChainedRemoveEntry, ChainedModifyEntry,
     ChainedModifyRDN,
     DSAReferral
     FROM DistributedOperations distributedOperations;

directorySystemAC
     APPLICATON-CONTEXT
          APPLICATION SERVICE ELEMENTS {aCSE}
          BIND          DSABind
          UNBIND        DSAUnbind
```

```
          REMOTE OPERATIONS {rOSE}
          OPERATIONS OF {
               chainedReadASE, chainedSearchASE,
               chainedModifyASE}
          ABSTRACT SYNTAXES {
               id-as-acse, id-as-directorySystemAS}
     ::={id-ac-directorySystemAC}

chained ReadASE
     APPLICATION-SERVICE-ELEMENT
          OPERATIONS {chainedRead, chainedCompare, chainedAbandon}
     ::= id-ase-chainedReadASE

chainedSearchASE
     APPLICATION-SERVICE-ELEMENT
          OPERATIONS {chainedList, chainedSearch}
     ::= id-ase-chainedSearchASE

chainedModifyASE
     APPLICATION-SERVICE-ELEMENT
          OPERATIONS {  chainedAddEntry, chainedRemoveEntry,
          chainedModifyEntry, chainedModifyRDN}
::= id-ase-chainedModifyASE

chainedRead                    ChainedRead      ::=1
chainedCompare                 ChainedCompare   ::=2
chainedAbandon                 ChainedAbandon   ::=3
chainedList                    ChainedList      ::=4
chainedSearch                  ChainedSearch    ::=5
chainedAddEntry                                 ChainedAddEntry
::=6
chainedRemoveEntry             ChainedRemoveEntry    ::=7
chainedModifyEntry             ChainedModifyEntry    ::=8
chainedModifyRDN                                ChainedModifyRDN
::=9

attributeError                 Attribute Error  ::=1
nameError                      NameError        ::=2
serviceError                   ServiceError     ::=3
dsaReferral                    DSAReferral      ::=9
abandoned                      Abandoned        ::=5
securityError                  SecurityError    ::=6
abandonFailed                  AbandonFailed    ::=7
updateError                    UpdateError      ::=8

END
```

(*Source:* ISO/IEC 9594-5: 1990, "Protocol Specifications.")

FIGURE 10.13 ASN.1 Definition of the Directory Service Protocol

"CORE ASE Wanna-bes"

The application tool kit is growing. A number of application service elements recently completed by ISO and CCITT offer capabilities that appear promising enough to speculate that they may ultimately become

members of the "core." Although it is beyond the scope of this book to address every application service element, there is a growing number of application service elements that one might classify as "core ASE wannabes." In the area of transaction processing, for example, there are three candidates. The OSI *commitment, concurrency, and recovery* (ISO/IEC 9804: 1990; ISO/IEC 9805: 1990), *distributed transaction processing* (ISO/IEC DIS 10026: 1992), and *remote database access* (ISO/IEC DIS 9579: 1992) services provide ways to associate a sequence of operations (a transaction or an atomic action) performed on remotely accessed data, such that either the entire sequence of operations must be performed on the data or the effects of all the operations must be undone. This "all or none" characteristic is referred to as *atomicity*. Assuring that the effects of the operations leave the data in either the original or the revised state (but no other) is called *consistency*. Another characteristic of processing a transaction is *data isolation*: from the time a transaction is initiated until it is completed, all data involved in the transaction may not be accessed by any other transaction.

A final characteristic of transaction processing is that once the data are changed, the changes endure failures of any subsequent transaction performed on the data. For example, if a transaction "foo" succeeds and results in a counter's value being changed from 1 to 2, the subsequent failure of a transaction "bar" cannot "undo" the effect of "foo" by leaving the counter with a value other than 2.

Commitment concurrency and recovery, transaction processing, and remote database access service elements are enabling vehicles for distributed-processing applications such as electronic banking (automated teller machines), point-of-sale inventory control, purchasing using debit cards, electronic brokering, and remote database access.

Early in the development of the OSI application layer, the terms *common application service elements* (CASE) and *specific application service elements* (SASE) were abandoned because no consensus could be reached on what criteria should be used to distinguish what was common from what was specific. This problem exists only if you persist in believing that the classification of an application service element is something static rather than dynamic. The true measure of whether an application service element is among the "core" set of application tools is how extensively it is used. Although the OSI Message Handling System and the Directory may be considered specific end-user applications today, in the future—as standards for office document management, banking, electronic data interchange, document filing and retrieval, and electronic library applications emerge—these, too, may become "core ASEs."

Conclusion

This chapter concludes the discussion of the application layer. It has examined the most frequently used application service elements—ACSE, RTSE, and ROSE—and illustrated how these ASEs provide services, individually and collectively, to the application service elements described in Chapters 7, 8, and 9. In the process, the authors have attempted to demonstrate the modularity of the OSI application layer and the flexibility afforded a user element (a specific application service) when a set of general-purpose mechanisms are made available to a developer of distributed application services.