DOC-CR-TP-86-002

METHODOLOGY AND GUIDELINES FOR APPLICATION PROTOCOL DEVELOPMENT

by
D.A. MacKinnon

Prepared for the Department of Communications
Under Contract Number OER85-Ø4Ø16

March 25, 1986
COMPUTER GATEWAYS INC.

EXECUTIVE SUMMARY

In conjunction with the development of the Reference Model for Open Systems Interconnection (OSI), ISO TC97 has been developing service definitions and protocol specifications for each of the seven layers identified in the model.

The interest in OSI is rapidly spreading beyond the bounds of ISO TC97. Both ISO TC68 (banking applications) and TC46 (bibliographic applications) have set up working groups to consider the application of OSI to their environments.

A characteristic of existing work on OSI is that it has been performed mainly by experts with a strong background in data communications and who have made large investments in learning OSI. This report presents a methodology and some guidelines to assist applications-oriented developers in the development of OSI-consistent Application Layer services and protocols.

This document is aimed principally at technical specialists who are involved in application development, but it is also of interest to managers and planners who have an interest in distributed processing. A third group of potential readers are those who have a general interest in OSI without any particular application in mind; for such readers, this document serves to integrate material from scattered sources.

A prerequisite to the development of OSI service and protocol specifications is an understanding of the upper layer architecture of OSI.

Each of the Session, Presentation and Application Layers is modelled in terms of a service provider which provides communications services on behalf of service users. Each layer has a different role. The Session Layer is responsible for organizing and synchronizing the dialogue between two systems and for managing the exchange of data. The Presentation Layer provides for the representation of information communicated between systems, that is it is concerned with the syntax and not semantics of interchanged information. The Application Layer contains all the functions which imply communication between open systems and are not already performed by lower layers.

Among the many concepts that are important to the application designer are those pertaining to associations, connections, application and presentation contexts, abstract and transfer syntaxes, principles of naming, OSI management and quality of service. Each of these is described in the context of the upper layer architecture.

The methodology for developing Application Layer services and protocols consists of four steps.

The starting requirement is an understanding of the distributed application for which services and protocols are to be developed. In particular, it is necessary to know which processes require interaction, in the OSI sense, with other processes. This information is used to determine the number of different protocols that are required and which of those are to be standardized.

With this knowledge, the first step of the methodology is to develop an OSI view of the application, in terms of service users and service providers. The relationships among the users are clarified and the principal conceptual data structures are identified. Any need for sub-layering is determined at this time.

The second step of the methodology is to identify the service elements relevant to the application. Service elements are the abstract elements of communications functionality; they are identified on the basis of operations on the conceptual data structures and other communications services. Also determined as part of this step is whether the application is connection-oriented, connectionless or store-and-forward in nature.

The third step of the methodology is to prepare a formal service definition according to the stated ISO guidelines. This includes a formal description of all service primitives, their parameters and their sequencing rules.

The fourth step of the methodology is to prepare a protocol specification. Such a specification includes an identification of the functions provided by the protocol, a listing of the services assumed from lower layers, a description of protocol behaviour and associated state information, a specification of the abstract syntax and associated transfer syntax(es), a set of protocol state tables and an identification of the application context(s). A formal specification of the protocol using an accepted formal description technique is recommended.

The entire methodology is illustrated using examples of existing ISO and CCITT services and protocols. In addition, a detailed example is provided of the application of the methodology to a typical banking application, namely interactive authorization.

It is important that validation of the protocol specification be performed during the design phase. A checklist is provided for performing this validation. Validation of protocol implementations is also required; the principal aspects of such protocol conformance testing are discussed.

A catalogue of services available to an application protocol is provided. This catalogue lists the capabilities available from the Session Layer, the Presentation Layer, the Common Application Service Elements, and existing Specific Application Service Elements. There is also a discussion of connectionless data transfer, security issues, multi-peer transmission and management information services. The application designer needs to be aware of all this information so that any new application can take full advantage of these capabilities and avoid duplication.

ISO has developed a number of concepts and tools which can be of assistance to an application developer. A discussion is provided of registration procedures, the Estelle and LOTOS formal description techniques, the Abstract Notation One, the ISO conformance testing methodology and some possible specification analysis tools.

Some of the concepts and terminology used in this document are not fully stable within ISO at this time. Therefore, it is expected that this document will require updating as work within ISO progresses.

Two areas where this methodology could usefully be applied in the banking area are point-of-sale and key management. Protocols are required in both of these areas on an urgent basis and the application of this methodology to this protocol development would be a valuable exercise.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Background

In conjuction with the development of the Reference Model for Open Systems Interconnection (OSI), ISO TC97 has been developing service definitions and protocol specifications for each of the seven layers identified in the model.

Within the Application Layer, there are two major thrusts in the international standardization effort. One is the development of common application services and protocols which are useful in a variety of applications. The other is the development of specific application services and protocols which are intended to satisfy particular requirements. ISO TC97/SC21 is currently standardizing three such protocols, one for file tranfer access and management, one for job transfer and manipulation and one for virtual terminal applications.

In addition to the work of TC97/SC21, OSI principles are being used by CCITT and by ISO TC97/SC18 for the development of electronic mail services and protocols.

The interest in OSI is rapidly spreading beyond the bounds of TC97. Both TC68 (banking applications) and TC46 (bibliographic applications) have set up working groups to consider the application of OSI to their environments.

A characteristic of existing work on OSI is that it has been performed mainly by experts with a strong background in data communications and who have made large investment in learning OSI. As OSI concepts become more widespread and is adopted by communities (such as banks and libraries) which do not have the same data communications background, then it becomes important that there exist a methodology to assist applications-oriented people in the development of OSI-consistent protocols.

This need has been recognized by TC68/SC5/WG4 (Applications of OSI in Banking) and a request for contributions has been sent to the member countries. It is felt that such a methodology would be of wide applicability among potential users of OSI services and would hasten the widespread adoption of OSI to satisfy business communications requirements.

During the previous contract performed for the Department of Communications, a document entitled "Evolving Toward OSI in Banking" was produced; in it, a brief description of such a design methodology ws provided, with an example drawn from the CCITT X.400 Recommendations.

1.2 Purpose

The purpose of this project is to expand the description of the methodology for developing applications services and protocols to produce a document in ISO Standard format for submission as a contribution to ISO.

The resulting document is intended to act as a "cookbook" for developers of Application Layer standards.

1.3 Approach

The bulk of this report is a document in ISO standard format which is to be submitted for consideration by ISO. This document is attached as Appendix A to this report.

This appendix covers four principal topics:

1) It provides an overview of architecture of the upper three layers of the OSI model. This establishes a context for the methodology, showing where new Application protocols fit in the overall OSI picture.

2) It describes the application development methodology in a step-by-step manner. Examples are provided of the application of the methodology to existing standards. In addition, a detailed explanation is provided of the development of a common banking application, namely interative authorization.

3) It provides a catalogue of existing Application, Presentation and Session services and their characteristics. The resulting catalogue acts as a standard "parts list" for application protocol design, helping the designer determine which aspects can be resolved by existing mechanisms and which aspects require specialized treatment.

4) It describes the set of available tools (such as formal description techniques and conformance test procedures) which can assist in successful protocol development.

2. CONCLUSIONS AND RECOMMENDATIONS

2.1 Conclusions

1. The nature of OSI is such that it is an area of endeavour that is constantly evolving. At present, many concepts are well understood, while others, notably relating to Application Layer structure and to the relationship between distributed application design and OSI protocol development, are only now being studied in earnest. It is natural then that a methodology document such as this be dynamic in nature. Modifications and extensions to this document should be anticipated as the OSI work matures.

2. This document has attempted to provide a consistent view of OSI concepts throughout. This was not always easy. It required that more general definitions be provided for some currently-used terms (e.g. service, service-provider) so that they could be applied to Application Layer services and protocols. It is evident that at present, there is considerable inconsistency in the use of terminology across the wide range of OSI documents currently in production and in use. This tends to confuse the reader and has the potential of leading to misinterpretation of the standards. Canada should strongly urge that the current effort at achieving consistency of terminology within the ISO committees working on OSI be given urgent priority.

2.2 Recommendations

There are a number of areas where work could profitably be done in support of OSI in banking. The following three are at this time likely the most valuable:

1. Ongoing Support for Methodology Document: If the current methodology document is to progress rapidly and effectively within ISO, then active support must be provided to explain and promote the methodology and to revise it in the light of comments from the international community. Canada should encourage the methodology work both within TC68 and TC97.

2. Point-of-Sale Example: If the methodology is to receive widespread acceptance, it is essential that examples be provided of its application. Point-of-sale processing is one such application. In particular, no standard currently exists for interactions between POS terminals and Card Acceptors. The development of an OSI-compatible protocol for this interaction would be a valuable exercise. The existing work done by the Australians in this area could serve as the basis.

3. Key Management Example: There is a recognized need for key management within both the banking and OSI communities. Work is underway within TC68 on key management but not in the OSI context. A liaison statement has been sent from ISO TC97/SC21/WG6 to TC68/SC5/W4 regarding cooperation on security aspects. Another group interested in security issues is ISO TC97/SC18/WG4 dealing with office communications; they are interested in developing an addendum to their electronic mail standard to incorporate security requirements. One of the most important requirements relating to security is key management. It is clear then than any work done the application of OSI to key management would be of interest to a wide community of standards making people.

   The base work for this effort could be ISO 8732, which is a standard for key management derived from ANSI X9.17.

APPENDIX A

METHODOLOGY AND GUIDELINES FOR APPLICATION PROTOCOL DEVELOPMENT

ISO/TC68/SC5/WG4 N_____
MARCH 17 1986

I S O
INTERNATIONAL ORGANIZATION FOR STANDARDIZATION
ORGANISATION INTERNATIONALE DE NORMALISATION

ISO/TC68/SC5/WG4

APPLICATIONS OF OSI IN BANKING

Secretariat: Canada (CBA)

TITLE: Methodology and Guidelines for Application Protocol Development

SOURCE: CANADA

TABLE OF CONTENTS

PREFACE

The Reference Model for Open Systems Interconnection (OSI) provides an architectural framework for the development of protocols for systems interconnection. This model describes the communications functionality of an open system in terms of seven layers. Each layer performs specific functions in support of the services it provides to the layer above it. The seventh and uppermost layer, the Application Layer, is an exception in that it does not provide services to a higher layer, but rather to the end user, which may be a human or an application process.

The Application Layer probably provides the greatest challenge to the protocol developer. If OSI is to achieve rapid and widespread acceptance, then it must be possible to develop application protocols for specific industries in a straightforward manner. The methodology described in this document attempts to assist the application protocol developer in this task.

This document may be of interest to a variety of readers. It is aimed primarily at technical specialists who are involved in application development, such as banking, and who are responsible for developing distributed applications incorporating OSI concepts. A background in electronic data processing (edp) and/or data communications will be helpful in understanding some of the abstract concepts which underpin OSI and the methodology described here. The technical reader who is not well versed in OSI will likely find all chapters useful. One who understands the OSI architecture could skip clauses 4 and 5 and proceed directly to the methodology clause (clause 6).

This document will also be of interest to managers and planners concerned with distributed processing, as it provides a framework for planning, developing and evaluating projects involving distributed processing. For these readers, the methodology clause (clause 6) will be of greatest interest. The other clauses may be of interest as sources of background information.

A third group of interested parties may be those who wish to better understand OSI concepts without any particular application in mind. For such readers, this document serves to integrate together material from scattered sources. Clauses 5, 7 and 8 will be of greatest interest. As the level of technical detail in Clause 5 varies, the reader who is interested primarily in basic concepts will derive the greatest benefit from subclauses 5.1, 5.2.1, 5.2.2, 5.3.1, 5.3.2, 5.4, 5.5.1, 5.5.6, 5.5.7, 5.5.8, 5.5.9 and 5.6.

All readers are assumed to have a basic grasp of the OSI Reference Model, to the point of understanding the concept of layering and the general purpose of each layer.

The intent of this document is that all or part of it be submitted for consideration by ISO TC68/SC5/WG4, ISO TC97/SC21/WG1 and ISO TC97/SC21/WG6 as a standard or technical report.

NOTICE TO READER

An effort has been made in the preparation of this document to make consistent use of OSI terminology. However, the many varied TC97/SC21 documents from which material has been extracted are not themselves consistent in their usage of terminology. This has caused difficulty for certain terms and concepts, particularly "service", "service provider", and "service element". The definitions used in this document are derived from those of ISO 7498, but extended in some cases to improve clarity and to make them applicable to the Application Layer.

All modified definitions or discussions of contentious points are flagged as such.

1. INTRODUCTION

The purpose of this document is to provide developers of industry-specific service and protocol standards for the Application Layer with a step-by-step methodology. This methodology will assist the developer in conceptualizing a distributed application in OSI terms, in identifying and formalizing the elements of service that a communicating system provides to the end user, in specifying the protocol actions and states required to support the service and in identifying which services of lower layers are required. As an illustration of the application of the methodology, the development of a protocol for interactive authorization of a bank card transaction is described.

The methodology is essentially a process of top-down decomposition and is influenced by the popular electronic data processing (edp) concept of abstract data types.

A pre-requisite to application protocol development is an understanding of the OSI Reference Model, and in particular the architecture of the upper three layers. A guide to this architecture is provided in Clause 5. The material in this clause is related to ongoing work within ISO TC 97/SC 21 dealing with upper layer architecture and various specific architectural questions such as Question 40, A General Model of Service.

In addition to the step-by-step description of the methodology, this document provides in Clause 7 a catalogue of existing Application, Presentation and Session services and characteristics. This information is helpful in ensuring that a protocol designer is fully aware of the capabilities and restrictions of these supporting services and therefore can properly assess the impact of these services on the design of an application protocol.

A set of development tools which can further assist the protocol developer are described in Clause 8. These tools include registration procedures, formal description techniques, specification analyzers, conformance testing procedures and an abstract syntax notation.

The emphasis in this document is on connection-oriented applications; this reflects the more mature state within ISO of concepts relating to connection-oriented applications vis-a-vis connectionless ones.

It is anticipated that this document will require periodic revision as architectural concepts relating to the Application Layer stabilize and additional tools are developed to assist the application development process.

2. SCOPE AND FIELD OF APPLICATION

This document describes a framework and a methodology for the development of industry-specific application service and protocol standards.

In particular, it provides:

a) a tutorial overview of the architectural principles affecting the upper three layers of the OSI Reference Model;

b) a step-by-step methodology which describes how to generate Application Layer service definitions and protocol specifications which conform to OSI architectural principles and that are in a format suitable for international standardization;

c) a description of existing Application, Presentation and Session Layer services and characteristics;

d) a description of protocol development tools that are in existence or under development.

This document is intended for use primarily by application standards developers, e.g. for banking or library communication, not directly involved in original OSI standards development. The approach described herein will also be acceptable for the development of standards within ISO TC97/SC21.

This document does not specify how to implement applications. It addresses only the development of communications-related specifications which are to form the basis for the design and validation of implementations.

*ISO DIS 8831 Information Processing Systems - Open Systems Interconnection - Job Transfer and Manipulation concepts and services.

*ISO DIS 8832 Information Processing Systems - Open Systems Interconnection - Specification of the Basic Class Protocol for Job Transfer and Manipulation.

ISO DIS 8822 Information Processing Systems - Open Systems Interconnection - The Presentation Service Definition

ISO 8326 Information Processing Systems - Open Systems Interconnection - The Basic Connection Oriented Session Service Definition.

*ISO DIS 8649/2 Information Processing Systems - Open Systems Interconnection - Definition of Common Application Service Elements - Part 2: Association Control.

*ISO DIS 8649/3 Information Processing Systems - Open Systems Interconnection - Definition of Common Application Service Elements - Part 3: Commitment Concurrency and Recovery

*ISO DIS 8824 Information Processing Systems - Open Systems Interconnection - Abstract Syntax Notation 1.

*ISO DIS 8825 Information Processing Systems - Open Systems Interconnection - Basic encoding rules for Abstract Syntax Notation 1 (ASN.1)

4. DEFINITIONS AND ABBREVIATIONS

4.1 Definitions

4.1.1 Abstract syntax: the aspects of the rules used in the formal specification of data which are independent of the encoding technique used to represent the data.

4.1.2 Application: an information processing task.

4.1.3 Application association: a cooperative relationship between two application entities.

4.1.4 Application-process: an element within a real open system which performs the information processing for a particular application.

4.1.5 Application entity: the aspects of an application-process pertinent to OSI.

4.1.6 Application context: an explicitly identified set of Application Service Elements, and options selected, available for use by the application entities during an application association, together with any other information necessary for their interworking.

4.1.7 Confirm (primitive): a service primitive issued by a service provider to complete, at a particular service-access-point, some procedure previously invoked by a request at that service-access-point.

4.1.8 Entity: an active element of a subsystem. An entity is local to a layer or sublayer.

4.1.9 Layer Service: the set of services available at the upper boundary of a layer. (Note: this is a new definition which attempts to distinguish between individual services of a layer and the global layer service).

4.1.10 Presentation context: an association of an abstract syntax with a compatible transfer syntax. The transfer syntax shall be compatible in the sense that it can be used to express all the information transfer requirements of the abstract syntax.

4.1.11 Request (primitive): a service primitive issued by a service user to invoke some procedure.

4.1.12 Service: a capability of a service provider which is provided at the upper service boundary to entities above that boundary. (Note: this is a variation of the definition in 7498 to include the Application Layer and CASE).

4.1.13 Service-access-point: the point where the services of a layer are provided by an entity of that layer to an entity in the layer above.

4.1.14 Service element: that part of an application entity which provides an OSI environment capability, using underlying services when appropriate. (Note: in this document, service element is equivalent to application-service-element as used in ISO 7498).

4.1.15 Service primitive: an elementary unit of interaction between a service user and a service provider

4.1.16 Service provider: an abstract machine which models the behaviour of the totality of the entities providing a set of services at the upper boundary of the service provider (Note: this is a variation of the definition in 85Ø9 to indicate clearly that a service provider does not provide a single service but a set of services).

4.1.17 Service user: an abstract representation of the totality of those entities in a single system that make use of a service through a single service-access-point.

4.1.18 Transfer syntax: those aspects of the rules used in the formal specification of data which embody a specific representation of that data used in the transfer of data between open systems.

4.2 Abbreviations

AA	-	Application Association
AE	-	Application Entity
ASN.1	-	Abstract Syntax Notation One
ASP	-	Abstract Service Primitive
ATM	-	Automatic Teller Machine
CASE	-	Common Application Service Element
CCR	-	Commitment, Concurrency and Recovery
CCS	-	Calculus of Communicating Systems
edp	-	electronic data processing
FDT	-	Formal Description Technique
FTAM	-	File Transfer, Access and Management
IA	-	Interactive Authorization
IUT	-	Implementation Under Test

JTM	-	Job Transfer and Manipulation
LM	-	Layer Manager
MIB	-	Management Information Base
MIS	-	Management Information Services
MHS	-	Message Handling Systems
MPDT	-	Multi-peer Data Transmission
MT	-	Message Transfer
OSI	-	Open Systems Interconnection
PCI	-	Protocol-control-information
PICS	-	Protocol Implementation Conformance Statement
POS	-	Point-of-Sale
PDU	-	Protocol Data Unit
PE	-	Protocol Entity
QOS	-	Quality of Service
SAP	-	Service-access-point
SASE	-	Specific Application Service Element
SMAE	-	System Management Application Entity
VT	-	Virtual Terminal

5. OVERVIEW OF THE UPPER LAYER ARCHITECTURE

5.1 Introduction

This clause provides a tutorial introduction to the upper three layers of the Reference Model. An understanding of this architecture is required for a proper appreciation of the methodology described in Clause 6.

This overview concentrates on the upper three layers as these are most relevant to application standards development. Of the seven layers defined in the Reference Model, the bottom four, namely Physical, Data Link, Network and Transport, deal with telecommunications functions and in principle are application independent. Thus, an application need not be aware of which Transport layer protocol class is being used during a connection, as long as the desired quality of service is being provided.

On the other hand, the upper three layers deal with processing functions related to a distributed application. These layers are inter-related, in that the choice of Session and Presentation Layer functions depends on the requirements of a particular application.

5.1.1 Layering

A key concept of the Reference Model is layering. The purpose of layering is to group together similar functions that are manifestly different from other functions in the process performed or the technology involved. This separation of function allows changes to be made within layers, e.g. in the choice of protocol, without affecting other layers, as long as the interfaces to adjacent layers are not affected.

In the case of the upper three layers, the separation of functionality is on the basis of dialogue management (Session Layer), syntax transformation (Presentation Layer) and functions not performed elsewhere (Application Layer). Overall control of communications is always vested in the Application Layer.

A more detailed look at the role of each layer is provided in the following.

5.2 Session Layer

5.2.1 Role of the Session Layer

The Transport Layer provides a simple, reliable, full duplex transmission service between two communicating systems. The Session Layer builds on this capability and provides to the Presentation Layer a set of services for organizing and synchronizing the dialogue between two systems and for managing the exchange of data.

5.2.2 Service Model of the Session Layer

The external, abstract view of the Session Layer is of a Service Provider which interacts with Service Users (SUs) at Service Access Points (SAPs), as shown in Figure 5.1. Each capability provided to the Service Users is called a service. A service is described in terms of a set of a group of abstract interactions which are called service primitives. SAPs are conceptually at the upper layer boundary, hence the Service Users are located in the Presentation Layer.

The Session service is in practice provided by Session Protocol Entities (PEs) making use of the services available from the Transport Layer through Transport SAPs. A Session Protocol Entity is the embodiment of Session protocol functionality within an open system. Figure 5.2 is equivalent to Figure 5.1, but provides more detail. Each of the possible mappings between Service User and Protocol Entity is illustrated. As can be seen from this diagram, each SAP defines a one-to-one relationship between a Protocol Entity and a Service User, although a PE or a SU may be associated with more than one SAP. Note that the Session Protocol Entities act as Service Users for the Transport Service.

5.2.3 Services of the Session Layer

In a connection-oriented environment, the services of the Session Layer can be classified into the following areas:

1. Connection Establishment which permits the establishment of a session connection and the negotiation of parameters related to the use of various functions.

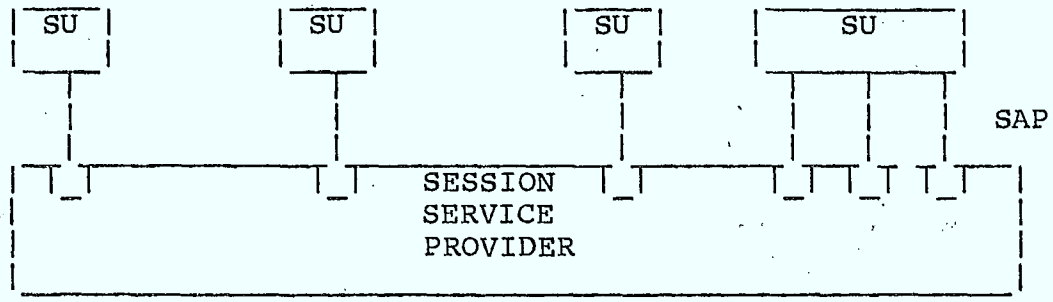


Figure 5.1 Model of Session Service

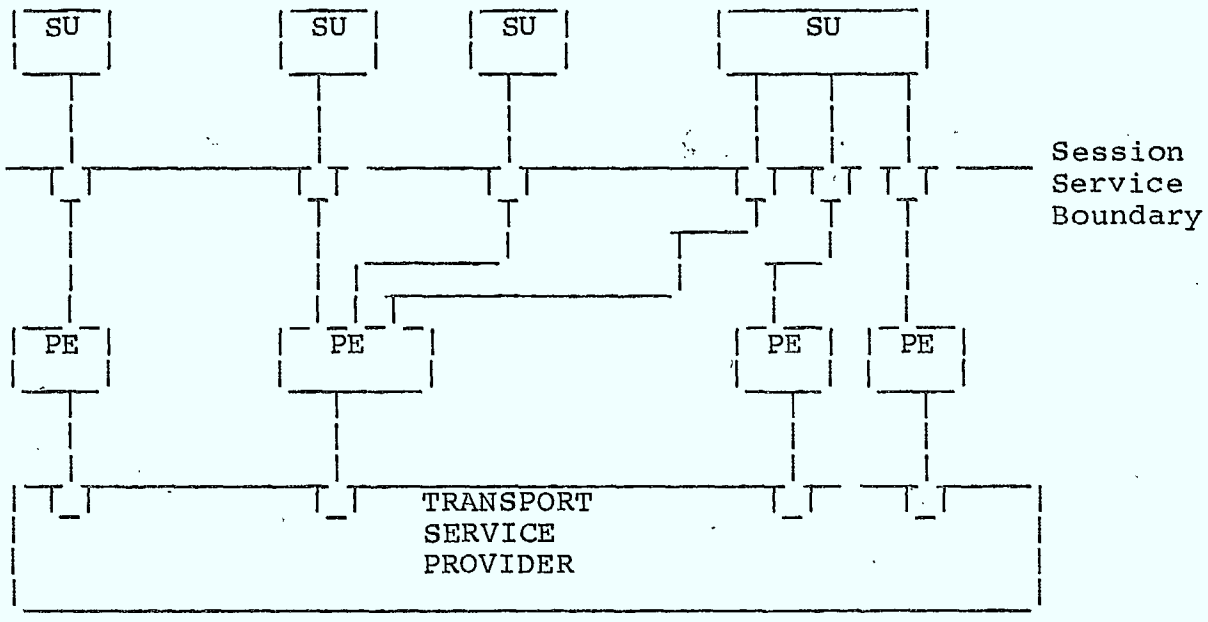


Figure 5.2 Exploded View of Session Service Provider

2. Connection Release which permits session disconnection. This disconnection can be:

   orderly, where all data already in transit is preserved; or

   abrupt, where the session connection is aborted and there may be loss of data.

3. Normal Data Transfer which permits the exchange of data with usual dialogue and/or flow control.

4. Expedited Data Transfer which permits the exchange of data where the exchange is not constrained by the dialogue control or flow control of normal data; in some instances, expedited data may bypass previously transmitted normal data.

5. Token Management which permits the requesting and transfer of tokens which control the exclusive right to exercise certain functions. For example, when token management is in effect, only the owner of the data token may initiate a normal data transfer.

6. Dialogue Control which permits a session to operate either in a two way alternate style, where only one of the systems (the owner of the data token) has the right to send data at any particular time, or a two way simultaneous style, where both systems are permitted to send data at any time. The two styles are called half-duplex and full-duplex, respectively.

7. Synchronization which permits the placement of synchronization marks in the data flow to mark and acknowledge identifiable points. Should an error be detected, this function makes possible the resetting of the session connection to a defined state and agreement on a resynchronization point (mark). Each mark has an associated serial number which is unique within a given session connection. The right to place such marks is controlled by the assignment of the major/activity token. Synchronization marks are of two types:

   Major marks which permit a clear delineation of the dialogue before and after the mark. The major mark must be confirmed by the receiver.

Minor marks which can optionally be acknowledged by the receiver.

8. Resynchronization which permits a "roll back" in the dialogue flow, i.e. designation of a resynchronization point, discarding of part of the data transfer and then a restart of the data transfer as though the data after the resynchronization point had not been previously sent. The resynchronization point in the dialogue is identified by a serial number associated with a (preceding) mark. It is not possible to resynchronize to a point earlier than the last confirmed major synchronization point. This style of resynchronization is "restart". A second style of resynchronization permits the present dialogue to be "abandoned" and a new serial number (i.e. one greater than any preceding serial number) to be used for the next synchronization point. A third style if resynchronization allows the serial number to be "set" to any value. Note that any semantics attached to serial numbers and the effect of resynchronization is left entirely to the user of the Session Layer.

9. Activity Management is an extension of the major synchronization concept. It provides the means to break a dialogue into discrete activities. Each activity can be regarded as a "separate" data transfer; however, the activity management function also has mechanisms which allow the identification of a particular activity, the transfer of data, interruption of the activity, and then resumption of the activity at a later time on the same or even on a different session connection.

10. Exception Reporting permits notification of unanticipated situations not covered by other services, e.g. protocol errors.

11. Typed Data permits the transfer of transparent user data independent of the token availability and position.

12. Capability Data permits the transfer of a limited amount of transparent data outside of an activity for special control purposes.

13. Negotiated Release permits one system to refuse a disconnect request and to continue in the data transfer phase.

5.3 Presentation Layer

5.3.1 Role of the Presentation Layer

The Presentation Layer provides for the representation of information that application-entities communicate between themselves.

The Presentation Layer is concerned only with the syntax, i.e. the representation of the data, and not with its semantics, i.e. their meaning to the Application Layer.

5.3.2 Service Model of the Presentation Layer

The model of the Presentation Layer is similar to that of the Session Layer. The only distinction is that the users of the Presentation Layer services are Application Layer entities and the Presentation Layer protocol entities are users of Session Layer services.

5.3.3 Abstract Syntax, Transfer Syntax and Presentation Context

The central function of the Presentation Layer is syntax transformation. The concepts of abstract syntax, transfer syntax and presentation context are key to the understanding of this function.

Formally, an abstract syntax is defined as the aspects of the rules used in the formal specification of data which are independent of the encoding technique used to represent the data. Informally, an abstract syntax can be viewed as describing the generic structure of data. For example, an abstract syntax could be defined in terms of a set of data type definitions.

Formally, a transfer syntax is defined as those aspects of the rules used in the formal specification of data which embody a specific representation of that data used in the transfer of data between open systems.

It is concerned with the way in which data is actually represented in terms of bit patterns while in transit.

For the purposes of transferring data between open systems, it is necessary to identify the abstract syntax being used and a transfer syntax that is capable of representing data values that may be generated using this abstract syntax. In general, there need not be a unique abstract/transfer syntax combination. It may be possible to support a specific abstract syntax with one or more different transfer syntaxes. Also, it may be possible to use one transfer syntax to support more than one abstract syntax.

Figure 5.3 illustrates the case where the Presentation Layer supports two transfer syntaxes for a single abstract syntax. Transfer syntax A uses character encoding for human readability while transfer syntax B uses bit-level encoding for transmission efficiency. The Application Layer provides identification of the abstract syntax being used and supplies data items encoded in the local system syntax. A syntax transformation module in the Presentation Layer accepts this information and generates encoded data items in one of the two transfer syntaxes. The choice of transfer syntaxes is negotiated by the communicating Presentation Layer protocol entities.

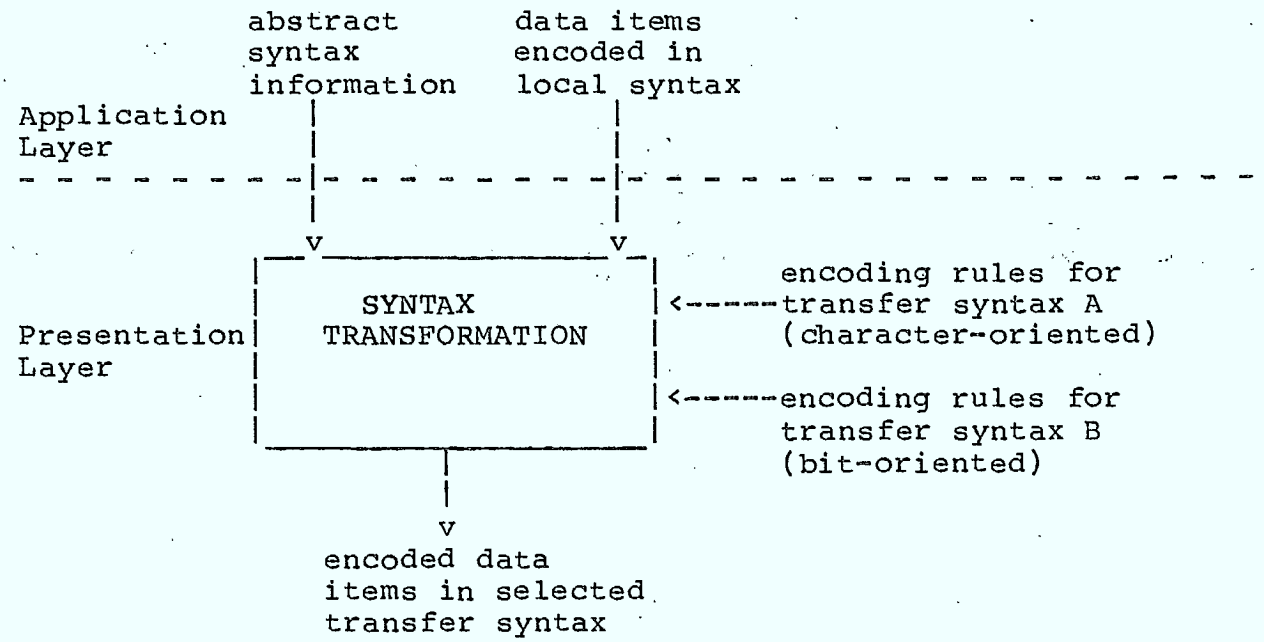


Figure 5.3 Illustration of Possible Mapping of an Abstract Syntax onto Multiple Transfer Syntaxes

A presentation context is defined as a particular abstract/transfer syntax combination that can be used for the transfer of data using the presentation service.

At least one presentation context is required to provide an application with a fully-defined environment for the transfer of data.

To define a presentation context, the service user identifies an abstract syntax while the service provider attempts to identify a transfer syntax which will both support the abstract syntax and be supported by the cooperating open systems. The service provider uses a negotiation mechanism to determine a mutually acceptable transfer syntax.

For the context definition mechanism to work it is necessary that both service users and the service provider have knowledge of the abstract syntax to be supported by the context. The presentation protocol assumes that abstract syntax specifications can be referred to by name. Moreover, the service provider requires knowledge of transfer syntaxes (or equivalently encoding rules) that may be associated with an abstract syntax. Again, the presentation protocol assumes these can be referred to by name.

In defining a presentation context, the negotiation mechanism enables the initiator to supply a list of transfer syntaxes, any one of which may be selected by the responder. The list of offered transfer syntaxes is in a preference order, to assist the responder in making a suitable selection in cases where more than one offered syntax is supported.

5.3.4 Layer Services

In a connection-oriented environment, the services of the Presentation Layer can be classified into the following areas:

1. Connection Establishment which permits the establishment of a Presentation Layer connection and the specification and selection of a set of initial transfer syntaxes.

2. Connection Release which permits disconnection of the Presentation Layer connection. This disconnection can be

   orderly, where all data already in transit is preserved; or

   abrupt, where the presentation connection is aborted and there may be loss of data.

3. Context Management which permits the definition and deletion of presentation contexts.

4. Data Transfer which permits five forms of data transfer:

   - data transfer subject to token control, depending on the selected session service;
   - data transfer with no token control;
   - expedited data transfer;
   - capability data transfer;
   - data transfer within "user data" fields of some services, such as P-CONNECT and P-ABORT;

5. Dialogue Control which permits access to Session dialogue control services.

5.4 Application Layer

5.4.1 Role

The essential purpose of OSI is to support distributed information processing. Such processing involves interworking among two or more application-processes. The Application Layer acts as the window between interworking application-processes which permits the exchange of meaningful information.

The Application Layer contains all functions which imply communication between open systems and are not already performed by the lower layers. These include functions performed by programs as well as functions performed by human beings.

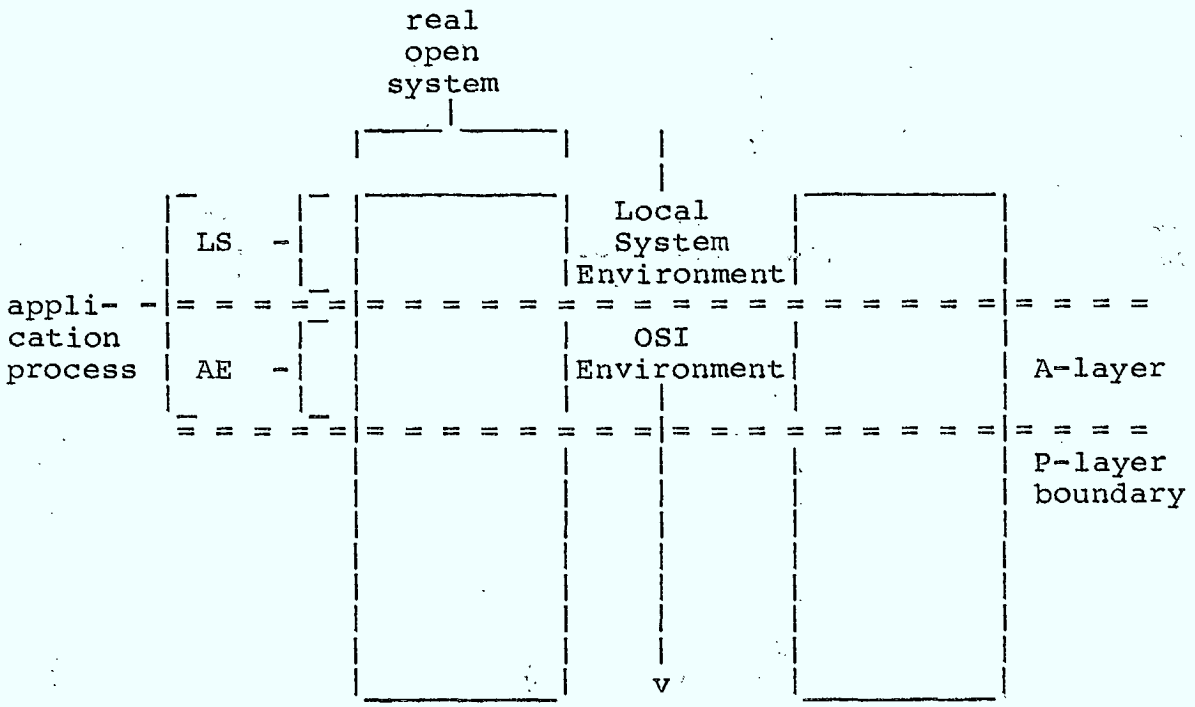
It is the only layer which directly provides services to application-processes and necessarily provides all OSI services directly usable by such processes.

5.4.2 Model

A distributed application may be modelled as a collection of application-processes each of which operates in two environments, as shown in Figure 5.4. The Local System Environment (LSE) is where local, communications-independent processing functions are performed; there is one LSE for each system participating in a distributed application. The Open Systems Interconnection Environment (OSIE) is the union of all communications functionality related to the distributed application. There is only one OSIE. Within a system, that part of the application-process which is concerned with communications, and hence is part of the OSIE, is called an application entity.

5.4.2.1 Service Elements

An application entity consists of a single user element and a set of application service elements. The term service element is used in this context to refer to a divisible part of the object that the entity represents; this is in contrast with the usage of the



Legend: A-layer = Application layer
P-layer = Presentation layer
LS = local system
AE = application entity

Figure 5.4 Local System and Open Systems Interconnection Environments

term service applied to other layers, which refers to functional interactions that can be described at the upper boundary of a layer.

Temporary Note: This interpretation of service element is consistent with its definition in ISO 7498, but differs from the usage in some ISO working documents where service element is equated with service.

The user element calls upon various application service elements to effect communication and application service elements call upon each other and presentation services to perform their functions. For convenience in description and standards development two categories of application service elements are recognized: Common application service elements (CASE) and specific application service elements (SASE). The acronyms CASE and SASE are each used to represent a set of service elements. A specific set of service elements is identified by a modifier, e.g. CASE kernel, FTAM SASE. ISO 7498 states that CASE provides capabilities that are generally useful to a variety of applications; and that a SASE provides capabilities required to satisfy the particular needs of a specific application. The distinction between the two categories is somewhat arbitrary and reflects an intuitive judgment that CASE provides services that will be used in most applications regardless of their nature and that a SASE provides services that are specific to a given application. Some SASEs are of broad utility and will be subject to standardization. Other SASEs will be specific to particular users and will not, in general, be the subject of standardization.

Application service elements receive service requests from each other and from the user element. Application service elements receive indications of services performed from each other and the attached presentation service access point (PSAP).

The user element is the original initiator of requests to the various application service elements and the ultimate recipient of responses from the various application service elements. Each application service element has a set of rules governing its use and use it makes of other application service elements, regardless of whether it is defined as a standard or privately designed. It is the responsibility of the element invoking or responding to application service elements to do so in a sequence that conforms to the applicable rules.

The Application Layer differs from other layers in that its services are not provided to a higher layer; the absence of an upper layer boundary means that there is no Service Access Point through which services are made available. Application service elements are considered an integral part of the application-process. Nevertheless, the service model used for the Session and Presentation Layers is still applicable. The service users are the user elements and the service provider is the totality of all service elements in the application entities which participate in the distributed application, plus all lower layers. In this case, the concept of SAP is not applicable, but it is possible to think in terms of a conceptual boundary between the user elements and the service provider. Figure 5.5 illustrates this.

Temporary Note: There is no agreed term for this conceptual boundary at this time; some ISO working documents use the term "service-element-access-point".

5.4.2.2 Application Context

An application entity includes all OSI-related aspects of an application-process. An application-process may have different communication requirements at different points in its processing. For example, it may need to interact with a remote terminal at one point and to transfer a file at another.

Related service elements within an application entity may be grouped together to form an application context. An application context is an explicitly identified set of application service elements, and options selected, available for use by communicating application entities during an application association, together with any other information necessary for interworking.

Temporary Note: Other definitions exist in various ISO working documents, but the essence remains the same in all cases.

In the case of simple applications, a single application context may be associated with an application entity. For example, an application entity that performed only file transfers would have a single application context. On the other hand, if an application entity supported both remote terminal access and file transfer, then two application contexts could be defined.

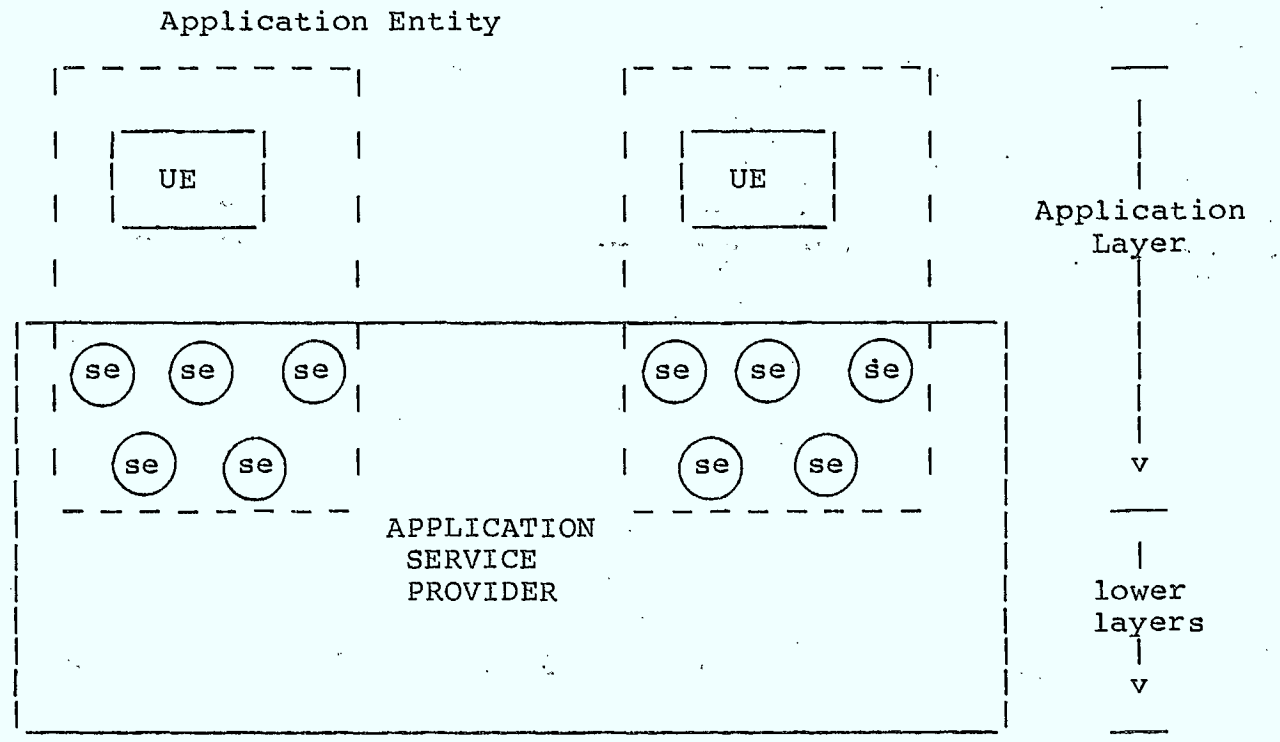


Figure 5.5 View of Application Service Provider

At any given instant during the lifetime of an application association, there is only one application context in use for a given direction of communication. This application context may be altered dynamically by mutual agreement of both application entities.

Temporary Note: The capability to dynamically alter the application context is not a capability of the current CASE kernel standard. This capability is defined in a proposed addendum to the CASE kernel to support context management.

5.4.2.3 Association Establishment

In order for meaningful interaction to be possible between application-entities, a cooperative relationship must exist. Such a relationship is termed an "application association" (AA). In a connection-oriented environment, application associations are established explicitly over a presentation connection. A CASE service element exists for this purpose.

The process of association establishment is initiated by the user element or some application service element in the initiating application-entity issuing a request for such an application association establishment to the A-ASSOCIATE service element. The A-ASSOCIATE service element then issues a P-CONNECT request across the attached presentation-service-access-point (PSAP). This P-CONNECT request results in protocol actions which trigger the occurrence of a P-CONNECT indication at the requested PSAP in the receiving system.

The A-ASSOCIATE service element in the application entity reached by this PSAP interacts with the service user to determine whether or not to accept the requested association and issues the appropriate response over the established presentation connection.

Each of the CASE Kernel functional units is disjoint. Among the parameters in an A-ASSOCIATE request is the one identifying the CASE functional units selected. Once an application association is established and this parameter agreed, this set of CASE is available for use over this association and is called the "default set of CASE" for this association.

Temporary Note: The following discussion on AE-Activations and the handling of multiple dialogues is not yet an agreed position within ISO TC 97/SC 21.

Only one dialogue is supported by a single application association. Therefore, when an application-process wishes to engage in multiple independent dialogues, then separate Application Entity Activations (AE-Activations) must be created to handle each association. In this case, each AE-Activation contains one full set of the service elements associated with that application entity. This is illustrated in Figure 5.6(a).

When an application process is involved in a cooperative relationship with two or more other application processes, then a separate association is established with each partner for each dialogue, but a single AE-Activation is created to handle all the related associations. In this case, each AE-Activation would manage more than one association. Figure 5.6(b) illustrates the case where three associations are managed by one AE-Activation.

All of the presentation service elements are available to the application service elements at the PSAP to which the application entity is attached. Since a single presentation connection supports a single application association, the presentation-connection-endpoint-identifier identifies, during the lifetime of that presentation connection, the application association and thereby the AE-invocation which is the source or sink for each presentation-service-data-unit crossing the conceptual interface at the PSAP for that connection. A parameter associated with each data value identifies the application service element that is the source or sink for this data value and through this mechanism the Presentation Service takes the responsibility for ensuring that the semantics of the data are transferred unambiguously.

5.4.2.4 Use of Services

A CASE makes direct use of the Presentation Service, and provides services to a SASE.

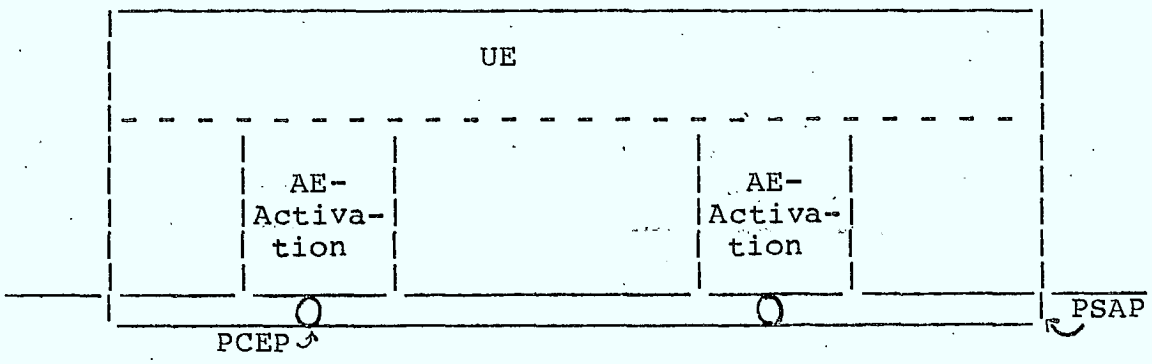
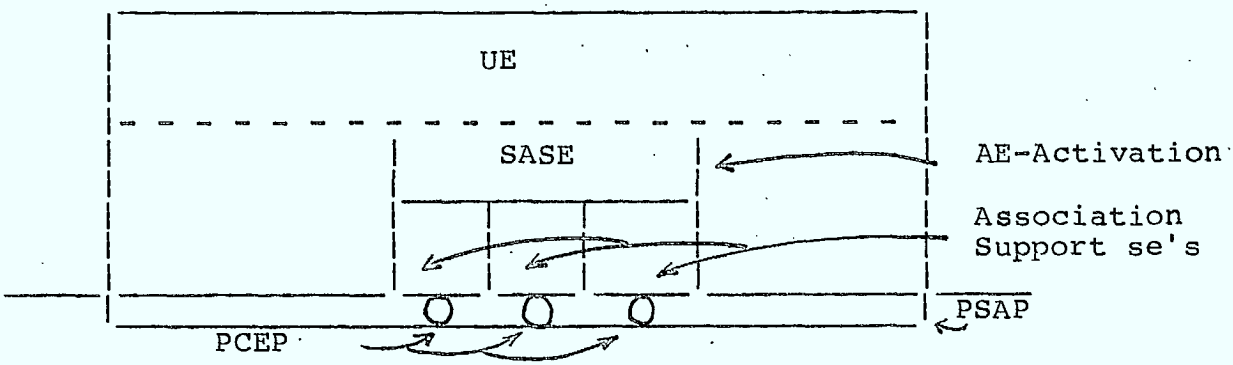


Figure 5.6 (a) Two Independent Dialogues with Separate AE-Activations



AE = Application Entity
PSAP = Presentation Service Access Point
PCEP = Presentation Connection Endpoint
SASE = Specific Application Service Element
UE = User Element

Figure 5.6 (b) Three Related Dialogues Managed by One AE-Activation

A SASE makes use of the services of CASE or of the Presentation Service interchangeably. A SASE may also make use of the services of other SASEs. At any given time, an AE-Invocation has a structure in the form as shown in Figure 5.7, with a User Element interacting with a controlling SASE, and with CASE and other SASE below. This structure is determined by the user element in its choice of controlling SASE.

Temporary Note: In Figure 5.7, direct access to Presentation services from a SASE is permitted. This may require change, if CASE incorporates these services as directly mapped services.

5.4.2.5 Relation to Application Context

Temporary Note: At present, the CASE kernel supports only one application context per association. The ability to switch contexts will be provided in the future via an addendum to the CASE kernel. The following discussion presumes the existence of that future capability.

Each SASE operates in a single application context associated with that SASE and that SASE alone. This application context continues to exist until the SASE completes its function.

If a SASE calls on the services of CASE, this does not affect the application context.

IF a SASE calls on the services of another SASE, the application context of the calling SASE is remembered, but the application context for the called SASE is used until that SASE completes its function, at which time the application context of the controlling SASE becomes effective again. The service elements for performing context switching are provided by CASE.

5.4.3 Modes of Communication

There are three possible modes of communication between open systems: connection-oriented, connectionless and store-and-forward.

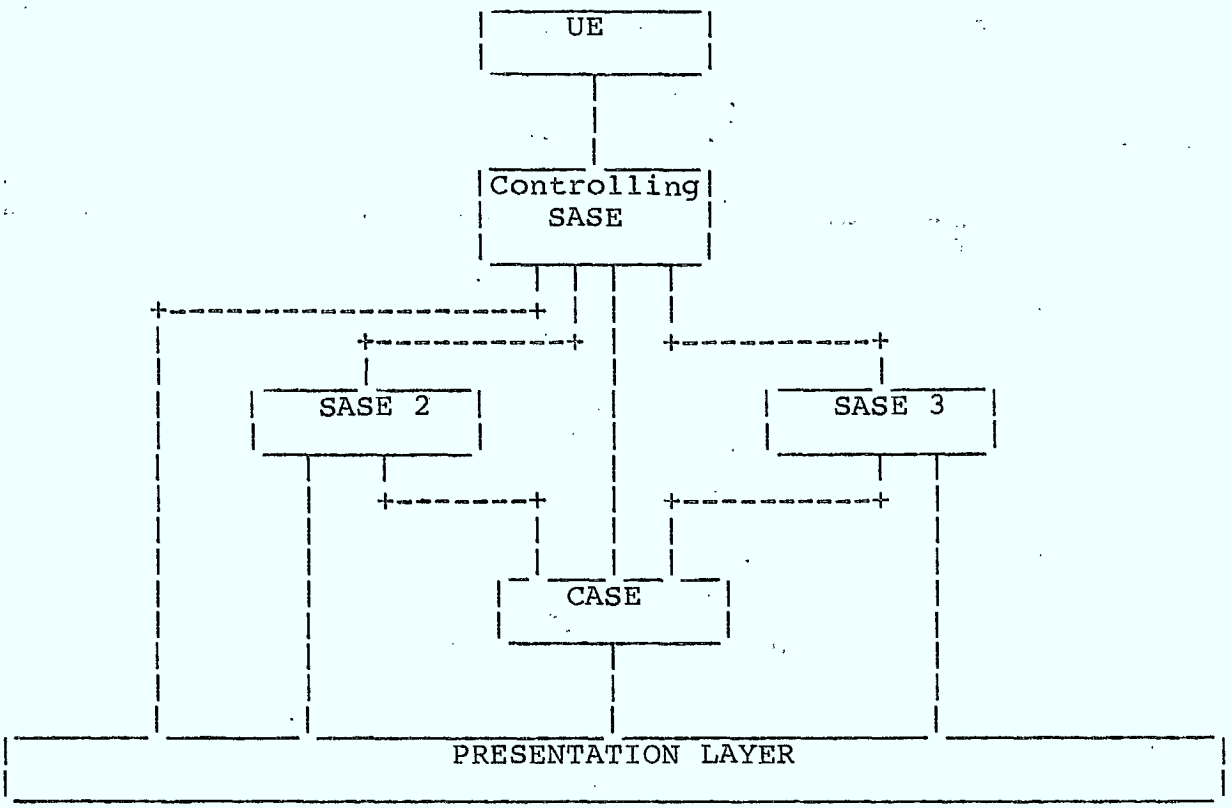


Figure 5.7 Structure of an Application-entity Activation

Temporary Note: At this time, ISO TC 97/SC 21 is working only on the connection-oriented and connectionless modes of communication. Work on store-and-forward communication is being performed by CCITT SG VII and ISO TC 97/SC 18, both in the areas of message handling (electronic mail).

For all three modes, interworking between application-entities depends on the existence of an application association. There are differences, however, in how an association is established.

In connection-oriented communication, an association is established explicitly each time interworking is to take place. It requires that the supporting layers 4 through 6 also be connection-oriented. Typically, the establishment of an association involves the establishment of connections in the supporting layers, usually layers 3 through 6. In situations where supporting connections already exist at one or more layers, then only those connections not already in place are built.

In connectionless and store-and-forward communication, there is no concept of end-to-end connections, so it is not possible to establish associations each time interworking is to take place. Instead, associations must be established a priori, either through the use of a directory service or through bilateral negotiation.

In connectionless communication, there is no concept of connection between any application entities. This mode is intended for the transfer of independent units of information, each of which is self-contained. The communications channel is potentially unreliable with the possibility that units of information will be lost, corrupted or received out of sequence. Architecturally, it is possible to include sequencing and error detection capabilities within individual layers, but at present, no standardized connectionless protocols include such functions. This mode of communication has the advantage that it has the least amount of processing overhead and therefore is the most efficient when sending small quantities of information. It is appropriate when transmission efficiency is of paramount concern and when transmission unreliability is either unimportant or acceptably low. An example of its use is for the remote acquisition of data samples that are to be averaged over time. It is also applicable for sending single units of information to

multiple destinations, particularly when the underlying communications medium supports broadcasting.

In store-and-forward communication, no end-to-end connections are established, but connections are established with intermediate systems which perform Application Layer relaying. This is illustrated in Figure 5.8. In effect, this mode is a mixture of the connection-oriented and connectionless ones. The highest level application protocol is connectionless on an end-to-end basis while a lower-level application protocol and all lower layers are connection-oriented on a hop-by-hop basis.

Thus, transmission reliability equal to connection-oriented communication is possible for each hop; however, the relay systems potentially can be failure points. This form of communication can be used when response time requirements are low or when an end system is not always available; in this case, the data can be sent to the final relay system for retrieval by the target system at its convenience.

Two implications of this mode of communication are that the amount of data that can be interchanged in one message is usually restricted due to limited buffer capacity of intermediate systems; and that there is no longer a one-to-one relationship between application association and traffic flow. Whereas in a connection-oriented environment, related traffic between application entities can be identified by connection-endpoint-identifier (which identifies the application association), in a store-and-forward environment, the existence of an association indicates only an ability to communicate; there may be many different traffic flows between two application entities taking place concurrently. An additional mechanism is needed to identify related messages, e.g. responses to requests.

5.5 General Concepts

5.5.1 Relationship to Implementations

The architecture described in this clause is conceptual in nature and in no way constrains implementations.

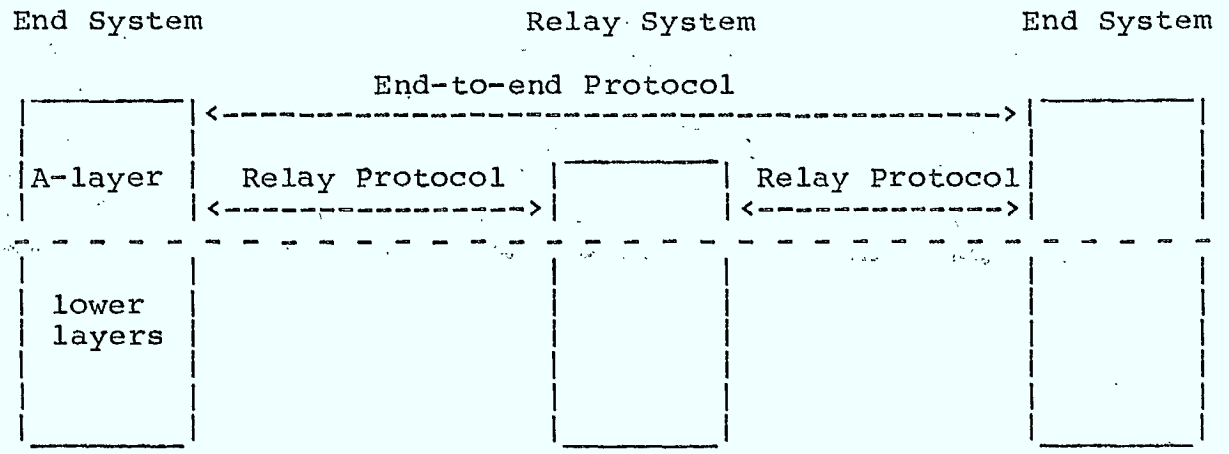


Figure 5.8 Illustration of Store-and-forward environment

In the OSI world, implementations are constrained only to the extent that implementations of protocols must conform to the conformance clauses stated in the corresponding standards. These conformance clauses specify only the externally-visible behaviour to which an implementation must adhere, and then only in terms of syntax and sequencing of protocol-data-units. There is no conformance to service definition standards.

This leaves the implementor considerable freedom.

In practice, service definition standards serve as a useful interface specification for layer implementations. With these standards, it becomes possible to develop bindings for commonly-used computer programming languages. A binding in this case is a mapping of a service primitive or sequence of service primitives into a language-specific procedure or function specification. The availability of such mappings in turn encourages the development of OSI software libraries. Although not eligible for standardization, language bindings for OSI services, if they become widely accepted and available, can help transform the aura of OSI software from "special purpose and therefore expensive" to "utilitarian and therefore inexpensive".

5.5.2 Relationship Between Association, Connection and Context

An application association is a cooperative relationship between two application entities.

There is a one-to-one mapping between an application association and a Presentation Layer connection, and similarly between a Presentation Layer connection and a Session Layer connection.

The establishment of a Presentation Layer connection is the first step in the process of establishing an application association. It permits identification of the cooperating application entities by their PSAP addresses. The second step is the exchange of A-ASSOCIATE protocol-data-units which serve to identify the application entities by title and to identify the application context that applies to the association.

The application context in turn identifies the set of service elements that are potentially available during the association. It acts as the frame of reference for successful interworking between the application entities.

5.5.3 Names, Titles and Addresses

In the OSI environment, names are linguistic constructs which denote communications objects. There are two classes of names: titles and addresses.

A title is a name given to an entity to identify it unambiguously. For example, each application entity has at least one title. A title is descriptive in nature only; it does not identify the physical location of an entity.

An address is a name used by a layer to identify a service-access-point through which access to an entity at the next higher layer can be obtained. An address serves as input to the routing function which determines how to access a given entity.

For an application entity, its application-entity-title at any instant in time is bound to the presentation-address of the PSAP to which the application entity is attached. This binding is held by the application directory service which provides a title-to-address mapping service. This directory service may be local to the end system or it may be a separate entity accessed by a specific protocol.

In the OSI environment, Network Layer addressing normally is used to identify end systems uniquely. Therefore, an addressing authority is needed to allocate unique Network Layer addresses throughout the entire OSI environment. For layers above the Network Layer, addresses are under the control of local system management. An address in this case is typically hierarchical in nature, consisting of a Network Layer address plus one or more selectors which identify a SAP uniquely within a layer of that system. In the case of a Session SAP address, it would consist of a Network address plus a Transport selector and a Session selector. A Presentation SAP address would add a Presentation selector to uniquely identify a PSAP among all the PSAPs associated with the Presentation entity identified by the Session SAP address.

For example, the process of establishing an application association may proceed as follows:

- the user element supplies the title of the application entity with which an association is desired;

- the PSAP of that application entity is obtained by inputing the application-entity-title to an Application Layer directory function, which provides a mapping from title to PSAP-address.

- another directory function, located in the Network Layer, provides the mapping between the destination NSAP-adress and routing information which is required to select a sequence of relay entities forming a path to the called NSAP.

- the entire PSAP-address is not conveyed in the protocol-control-information of the Presentation Layer. Instead, it is decomposed in the initiating open system into its constituent parts namely:

  . Presentation-selector
  . Session-selector
  . Transport-selector
  . NSAP-address

  and these are conveyed, as addressing information in the protocol-control-information of the Presentation, Session, Transport and Network Layers, respectively. At the recipient open system, they are composed again into:

  . the NSAP-address
  . the pair (NSAP-address, transport-selector) = TSAP-address
  . the pair (TSAP-address, session-selector) = SSAP-address
  . the pair (SSAP-address, presentation-selector) = PSAP-address

  to refer to, respectively:

  . a transport entity
  . a session entity
  . a presentation entity
  . an application entity.

If it is necessary to confirm that the required application-entity is still attached to the PSAP identified by the presentation-address, then the application-entity-title can be passed as part of the A-ASSOCIATE protocol-control-information.

5.5.3.1 Naming Authorities

A name singles out a particular object from among a collection of objects. Names are unambiguous, but not necessarily unique. A name must certainly be unambiguous, that is, denote just one object. However, a name need not be unique, that is, be the only name that unambiguously denotes the object.

The requirement for unambiguity of names implies some authority with control over the allocation of names. ISO has proposed a hierarchical naming convention such that different naming authorities would have responsibility for naming domains, corresponding to different levels in the naming hierarchy.

The allocation of naming authorities remains to be done.

Every local system manager will be a naming authority for the naming domain consisting of SAP address selectors within a given end system.

5.5.4 Embedding of Protocol Data Units (PDUs)

At any layer, the protocol data units exchanged between peer entities has two components: protocol-control-information relating to the operation of the protocol and user data supplied by the service user. This user data is not meaningful to the protocol entities and is delivered intact to the recipient service user.

This fundamental concept of embedding is illustrated in Figure 5.9 for connection establishment. In this example, a user wishes to transfer a file using the file transfer SASE. The first step is to initialize the file transfer protocol. This results in the generation of file transfer protocol-control-information (PCI) which is supplied as user data to an A-ASSOCIATE CASE request. CASE adds its own PCI to form a CASE A-ASSOCIATE PDU. This PDU in turn becomes user data of a P-CONNECT request. The resulting Presentation PDU then becomes user data of an S-CONNECT

request. The resulting S-CONNECT PDU is sent as user data of a T-DATA request after a Transport connection has been established.

The example above illustrates the case where the Application association and the Presentation and Session connections are established at the same time. In this situation, the failure of connection establishment in any layer causes failure in all of the upper three layers.

It is not necessary that the upper layers establish connections in unison. It is also possible to establish layer connections sequentially. In this case, the connect PDU of one layer is sent as user data of a DATA PDU of the lower layer once the lower layer connection is established; this corresponds to the behaviour of the Session Layer relative to Transport connection establishment.

5.5.5 Directly Mapped Services

Services provided by one layer may be offered by the next higher layer to its user without any additional value. These services are called directly mapped services.

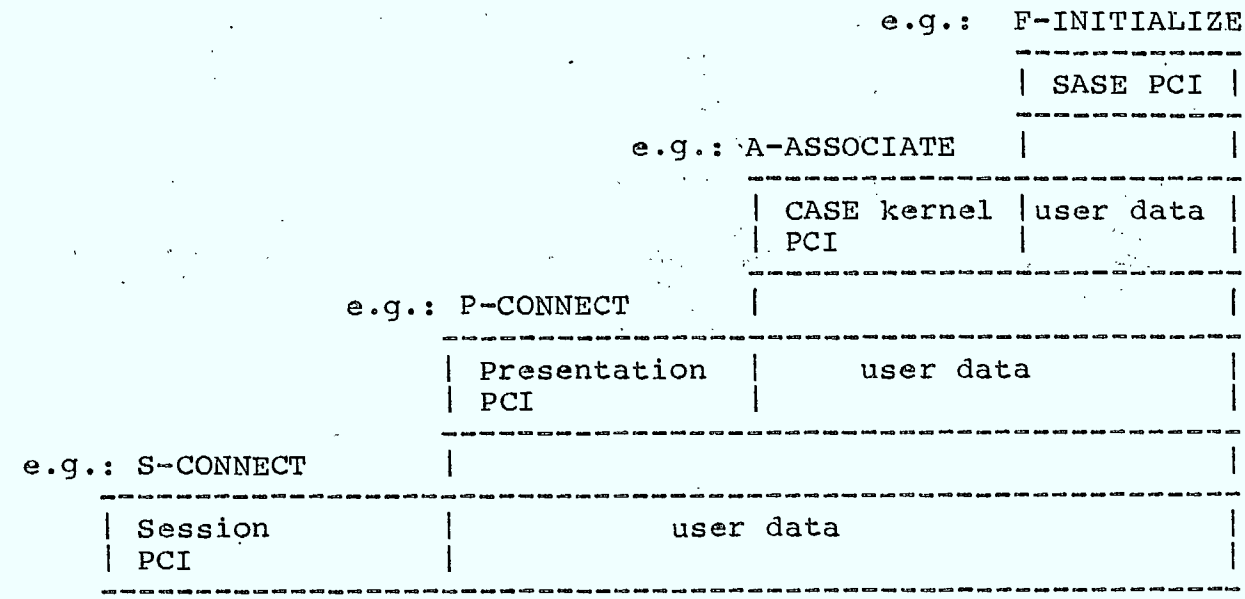
However, the invocation of some directly mapped services may cause a change in the state of a service provider even though it may not generate any additional PCI to support the service. Those directly mapped services which do not cause any change in the state of a service provider are called pass-through services.

5.5.6 Management

Within OSI, there are requirements for the planning, organizing, supervision and controlling of the communications aspects of a distributed information processing system.

These management aspects are decomposed into two sets of activities:

a) OSI management, related to the interworking of OSI Management-application-processes across open systems. It provides the means by which management information is distributed between open systems, management information is accessed by remote open



PCI = Protocol Control Information

Figure 5.9 Example of Embedding of Protocol Data Units

systems and the execution of management processing is controlled. An example of the latter capability is initiation of accounting activities.

b) open-system-management, related to the management information needs of individual open systems. It provides the means by which management information is stored and retrieved and management information is exchanged among the layers.

Architecturally, management functionality is divided into system management and layer management.

5.5.6.1 System Management

System Management in the OSI environment is achieved through a set of application processes running in open systems, which communicate with each other to support the management activities described below:

There are five different categories of system management:

1. Configuration Management: determination and control of the logical and physical configuration of the system;

2. Performance Management: control and assessment of the performance of the individual end systems and relay systems of a network and of overall network operation;

3. Fault Management: detection, diagnosis and reporting of failures;

4. Access Management: control of access to resources and processes;

5. Accounting Management: tracking and reporting usage of the resources;

A system management process, like any other OSI application process, consists of two parts: one, the application entity, which resides in the Application Layer, deals with communication aspects and is relevant to OSI; the other handles local system management which is outside the scope of OSI.

The system-management-application-entity (SMAE) handles all inter-system communication, including communication relating to coordination of system managers and to coordination of layer managers (see below).

Local system management has responsibilities for:

- starting up a system
- serving as the intermediary for the exchange of management information between layers;
- initiating the layer manager for each individual layer
- serving as the manager of management information that is common to several layers or that is supplied externally.

5.5.6.2 Layer Management

In keeping with the general OSI principle that each layer is independent of all others, each layer has its own management function. Each OSI system has a layer manager (LM) for each layer. Layer management is the collection of LMs for a particular layer.

A layer manager has the following roles:

1. it serves to coordinate the activities of the entities within a layer. This is principally the activation and deactivation of entities as needed.

2. it acts as the window to system management for layer entities. For example, when an entity needs access to an operating parameter stored in the general management information base (MIB), e.g. a timeout interval, the layer manager will retrieve the information on behalf of the entity. Also, when entities in different layers need to communicate, they must do so via the local system manager. (Direct communication between layers for management purposes is forbidden in order to prevent dependencies which could violate layer independence).

3. it manages the layer. This is done in conjunction with both system management and peer LMs. Note that LMs within a layer communicate via the system manager, i.e. via an Application Layer protocol.

To gain access to the SMAE, an LM must act through the local system manager. This is shown in Figure 5.10.

5.5.6.3 Directory Service

An important aspect of OSI management is the directory service. This service provides a common method for the storage, manipulation and retrieval of information concerning communications. It fulfils two essential needs of OSI networks:

1. to allows clients (i.e. users and applications) to rely on user friendly names;

2. to make the OSI network "self-configuring" in the sense that addition, removal, and changes in the physical location of objects do not affect network operation.

The directory service provides its clients with dynamic binding of names to other names, names to groups of names and properties of objects to the names of objects with those properties.

An example of the name-to-other-name binding service is the application-title to SAP address mapping discussed under naming and addressing. This is analogous to a "white pages" directory.

An example of the name-to-list-of-names binding service is a mailing list for an electronic mail application.

An example of the property-to-set-of-names binding service is to find the names of all systems which support a particular protocol. This is analogous to a "yellow pages" directory.

As with other management services, the directory service lies within the Application Layer.

5.5.7 Registration

For two open systems to interwork, it is not sufficient that they both implement the same set of protocols. There are a number of other aspects of which the two systems must be cognizant for successful distributed processing to be possible.

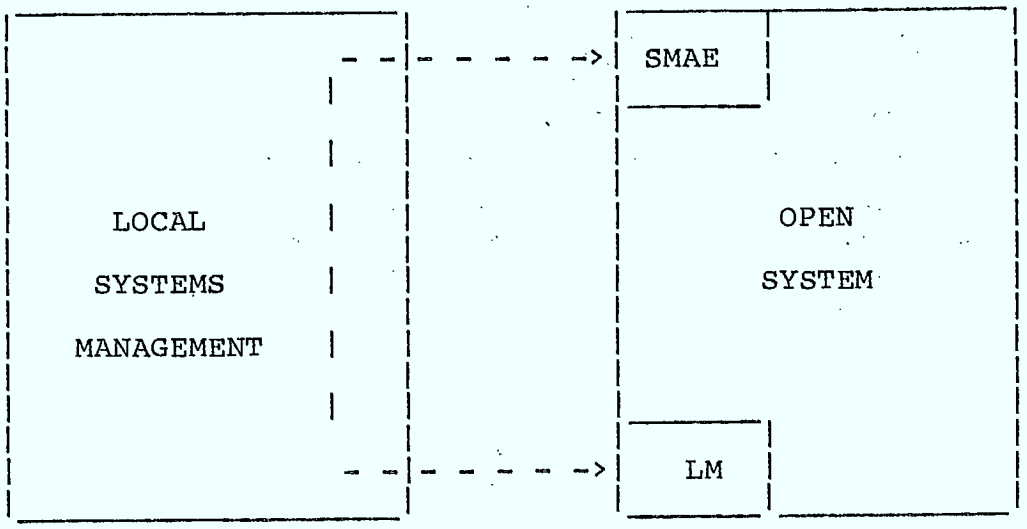


Figure 5.1Ø Interactions Among System Management Entities

One obvious example is the title or address of the destination application entity that an initiating entity wishes to reach. While the structure of such names may be standardized, the actual names are allocated by a naming authority. This allocation is a form of registration.

Registration is the act of allocating an unambiguous name for a structured amount of information. A registration authority is typically assigned the responsibility for ensuring this unambiguity and for recording the registered information. It may also be responsible for ensuring completeness and accuracy of the information to be registered. When ISO acts as a registration authority, it specifies the formal procedures that an applicant must follow.

When information to be registered is not relevant to all of OSI, for example when a set of names are relevant only to the banking industry within a country, then the registration authority may be assigned to a different body.

Another type of information for which registration is required is application context names. Knowledge of application context is crucial to proper interworking of two application entities, and it is therefore important that application context names are known to the interested parties and are unambiguous. This is achieved by registering such names with a registration authority who is responsible for ensuring unambiguity of context names and for making public the set of registered contexts.

Registration authorities are also necessary for registering abstract and transfer syntaxes as there may be many more such syntaxes than are standardized by ISO.

For a particular SASE, there may be other requirements for a registration authority. For example, the Job Transfer and Manipulation (JTM) service has a requirement for the registration of document types that can be interchanged using JTM.

5.5.8 Security

The objective of OSI is to permit the interconnection of heterogeneous computer systems so that useful communication between application processes may be

achieved. At various times, security controls must be established in order to protect the information exchanged between the application processes. Such controls should make the cost of illegally obtaining or modifying data greater than the potential value of obtaining or modifying the data.

ISO has developed in 7498/2 an architectural framework for the provision of security features within OSI. No security functions have been incorporated yet into standard protocols.

A number of security features can optionally be provided within the framework of the OSI Reference Model:

1. peer entity authentication: this capability is provided at the establishment of a connection in order to provide a high degree of confidence that the connection has been established with the addresed peer entity (and not with an entity attempting a masquerade or a replay of a previous connection establishment).

2. access control: this capability provides protection against unauthorized use of the resources accessible via OSI. This protection capability can be applied in common to a group of entities (often referred to as a closed user group) or on an individual entity basis.

3. data confidentiality: this capability provides for the protection of data from unauthorized disclosure. Confidentiality can have the following features:

   a) confidentiality of all user data on a connection;

   b) confidentiality of user data in a single connectionless service-data-unit;

   c) confidentiality of selected fields within the user data of a service-data-unit; and

   d) traffic flow security, i.e. protection of the information which might be derived from observation of traffic flows.

4. data integrity: this capability detects active attacks and it may take one of the following forms:

   a) integrity of a single connectionless service-data-unit. This may take the form of determination of whether a received service-data-unit has been modified;

   b) integrity of selected fields within a single connectionless service-data-unit. This takes the form of determination of whether the selected fields have been modified;

   c) integrity of selected fields transferred over a connection. This takes the form of determination of whether the selected fields have been modified, inserted, deleted or replayed;

   d) integrity of all user data on a connection. This service detects any modification, insertion, deletion or replay of any service-data-unit of an entire service-data-unit sequence (with no recovery attempted);

   e) as for (d) but with recovery.

5. data origin authentication: this capability provides assurance that the data source is the one claimed.

6. non-repudiation: this capability may take one one or both of two forms:

   a) the recipient of data is provided with proof of the origin of data which will protect against any attempt by the sender to falsely deny sending the data or its contents; and/or

   b) the sender is provided with proof of delivery of data such that the recipient cannot later deny receiving the data or its contents.

A variety of mechanisms have been identified to provide some of the capabilities described above. These include encipherment, digital signature mechanisms, access control mechanisms (e.g. passwords), data integrity mechanisms (e.g. use of redundancy checks), traffic padding (i.e. sending spurious data to disguise the length and number of transmitted service-data-

units), routing control (e.g. selecting only secure communication paths), notarization (i.e. use of a trusted third party as a relay), ensuring trusted functionality of software and hardware, incorporation of event detection and handling procedures (e.g. local and remote reporting of unusual events) and provision for audit trails (e.g. logging all events).

The above security services may be provided by different layers according to circumstance. The following table from 7498/2 indicates the layers of the Reference Model in which particular security services can be provided.

Table 5.1 Matrix of Security Services and Layers

SERVICE	LAYER 1	2	3	4	5	6	7*
Peer Entity Authentication	N	N	Y	Y	N	Y	N
Access Control	N	N	Y	Y	N	Y	Y
Sequence Confidentiality	Y	Y	Y	Y	N	Y	N
Connectionless Confidentiality	N	Y	Y	Y	N	Y	N
Selective Field Confidentiality	N	N	N	N	N	Y	N
Traffic Flow Security	Y	N	Y	N	N	N	Y
Connection Integrity (no recovery)	N	N	Y	Y	N	Y	N
Connection Integrity (recovery)	N	N	N	Y	N	N	N
Selective Field Connection Integrity	N	N	N	N	N	Y	N
Connectionless Integrity	N	N	Y	Y	N	Y	N
Selective Field Connectionless Integrity	N	N	N	N	N	Y	N
Data Origin Authentication	N	N	Y	Y	N	Y	N
Non-Repudiation (origin)	N	N	N	N	N	Y	N
Non-Repudiation (delivery)	N	N	N	N	N	Y	N

Legend: Y - Yes, service can be provided in this layer
N - No, service shall not be provided in this layer
* - While application entities are restricted in the security services which they can provide, the application-process, as a local matter, may itself provide security services.

5.5.9 Quality of Service

In the OSI environment, it is possible for an end user to specify its requirements in terms of the quality of service (QOS) it needs for communication.

In a connection-oriented environment, these requirements are stated at the time of connection establishment, and if accepted by the service provider, then are expected to be maintained throughout the lifetime of the connection; if a service provider knows that it is unable to satisfy these requirements, then it refuses the connection.

In a connectionless environment, quality of service is specified each time a data unit is to be transferred.

While it is the end user or application entity who typically states QOS requirements, it is the Session Layer which is responsible for satisfying them. This it attempts to do with the support of the Transport and Network Layers.

Quality of service is typically stated in terms of a set of parameters. Some of these parameters can be classified as performance-related while others deal with non-performance issues.

The performance-related QOS parameters are further divided into speed-related ones and accuracy/reliability-related ones.

Speed-related QOS parameters are concerned with how fast activity can take place and typically set limits on the maximum acceptable values. For example, the Session Connection Establishment Delay parameter states the maximum acceptable value for the time taken for a successful connection establishment to take place.

Accuracy/reliability-related QOS parameters are concerned with the possibility of errors and generally state the maximum acceptable probability of failure. For example, the Residual Transfer Rate parameter states the acceptable ratio of incorrect, lost and duplicate units of user data to the total user data traffic volume in a measurement period.

The non-performance-related QOS parameters deal with a variety of issues:

1. the protection parameter states requirements for protection measures such as those discussed under Security previously;

2. the priority parameter specifies the relative importance of a session connection in case QOS must be degraded or resources must be reclaimed;

3. the extended control parameter allows Session Service users to make use of the resynchronize, abort, activity interrupt and activity discard services when normal data flow is congested. This is useful if the expedited data functional unit is not available.

4. the optimized dialogue parameter permits concatenated transfer of certain session service requests to improve transmission efficiency.

Table 5.2 lists all the available Session QOS parameters.

Table 5.2 Summary of Session Quality of Service Parameters

PERFORMANCE-RELATED	
SPEED:	Session Connection Establishment Delay Session Connection Release Delay Throughput Transit Delay
ACCURACY/RELIABILITY:	Session Connection Establishment Failure Probability Session Connection Release Failure Probability Residual Error Rate Session Connection Resilience Transfer Failure Probability
OTHER	
	Extended Control Session Connection Protection Session Connection Priority Optimized Dialogue Transfer

5.6 Relationship Among Layers

5.6.1 Session and Presentation Layers

The partitioning of functions into the Session and Presentation Layers represents a separation of concerns from the viewpoint of OSI protocol design; it does not reflect a hierarchical relationship between such functions. Consequently both Session and Presentation Layer services are visible to a user of the presentation services.

The complementary nature of the functions provided in these layers, coupled with the layered approach to protocol design, requires that Presentation Layer standards make Session Layer functions available to a presentation service user in an essentially unmodified form. It is this complementary nature of the layers that gives rise to the concept of directly-mapped services.

5.6.2 Presentation and Application Layers

The principal function of the Presentation Layer is to perform syntax transformation. For this to be possible, the Presentation Layer must be aware of and must understand the abstract syntax(es) in use by the Application Layer on a given connection. Thus, the Application Layer is responsible for identifying the abstract syntax(es)

The use made by the Application Layer of the Presentation Layer services (excluding directly-mapped Session Layer services) is very much dependent on the variety of syntaxes that the Application Layer may use on a given Presentation connection.

In the simplest case, only one abstract syntax is used by the Application Layer and the local representation of that syntax corresponds to the transfer syntax. In this case, no data transformation functionality is required of the Presentation Layer; its role is reduced to connection establishment and release, and acting as a conduit for Session Layer services.

More typically, more than one abstract syntax is required for an application. Normally, one syntax would be used for CASE and another for each SASE that is invoked by the application. Depending on the selected abstract syntaxes, negotiation of transfer syntax may or may not be needed.

When more than one transfer syntax can be used with a given abstract syntax, then the selection of transfer syntax by the Presentation Layer may be influenced by local management considerations, e.g. a requirement for lowest cost transfer would encourage the choice of a transfer syntax which provided data compression.

Hence, the functionality of the Presentation Layer for any given connection is dependent on the requirements of the Application.

5.6.3 Application and Session Layers

The Session Layer provides a variety of functions that typically are not in use together on a connection. Related functions are grouped into functional units which must be selected at time of Session connection establishment. The choice of which functional units to select depends on the requirements of the Application.

The Session Layer provides many services that are useful for managing the dialogue between two application entities. It is very important to realize that these services are only tools; the Session Layer itself does not make any decisions about what to do in particular situations, e.g. whether a resynchronization should be of type "restart" or type "set". It simply executes the function requested by the Session user.

An important responsibility of the Application Protocol designer is determining how best to make use of the available Session services. This issue is discussed further in Clause 7.

6. METHODOLOGY FOR THE DEVELOPMENT OF SASE STANDARDS

6.1 Introduction

This clause describes the steps in the development of OSI-compatible standards for Specific Application Service Elements. For each step, the nature of the information to be supplied, the level of detail and the presentation format is defined.

Also, a description is provided of the application of each step of the methodology to the development of an example SASE. The example is interactive authorization, as described in ISO 8583.

This methodology is consistent with the ISO guidelines as stated in TR 85Ø9.

****
*
NOTE: Material that is tutorial in nature, e.g. related to the example, is set between rows of asterisks.
*
****

6.2 Objective of Methodology

The objective of the methodology described here is to develop service and protocol standards which are compatible with the OSI Reference Model and which are consistent in organization and preseentation style to OSI standards developed by ISO TC97/SC21.

It is not an objective of this methodology to describe how to design distributed applications. Such applications include an information processing component and a communications component. Only the communications component is of interest here. It is recognized that the communications component cannot in general be designed in isolation. An understanding of the overall application is required. However, there are many possible approaches to the design of an overall application and no consensus exists on the "best" one to use. The methodology described here identifies only what information must be known a priori before service and protocol standards development can begin. How this information is obtained is outside the scope of this document.

The methodology takes a top-down, data-driven approach based on the popular edp concept of abstract data types.

****
*
*

6.3 Overview of Interactive Authorization Example

Interactive authorization (IA) is an application related to financial transactions involving bank cards. It is one of the functions supported by ISO 8583. A high level model for bank card-related processing recognizes that four application-processes may be involved: the Cardholder, Card Acceptor, Acquirer and Card Issuer. These application-processes and their interactions are illustrated in Figure 6.1.

The Cardholder is the initiator of a financial transaction; it typically is a person. The Card Acceptor may correspond to a clerk in a store, to a Point-of-Sale (POS) machine or to an Automatic Teller Machine (ATM). The Acquirer is typically the merchant's financial institution and may act in certain cases as the agent of the Card Acceptor for interactions involving a Card Issuer. The Card Issuer is the financial institution who issued the bank card to the Cardholder and has the authority to accept or refuse transactions initiated by a Cardholder.

In this environment, the purpose of interactive authorization is to obtain approval or guarantee for a financial transaction to proceed. An authorization request flows from the Card Acceptor to the Acquirer, and then to the Card Issuer. The response flows in the opposite direction along the same path as the request.

An authorization is not intended to convey sufficient data to permit the application of a transaction to the Cardholder's account for the purpose of issuing a bill or statement. An important aspect of this application is its real time nature, with a strict upper limit on response time between issuance of a request for authorization and receipt of a response. An example of an authorization is a request for approval of a credit card purchase.

*
*
****

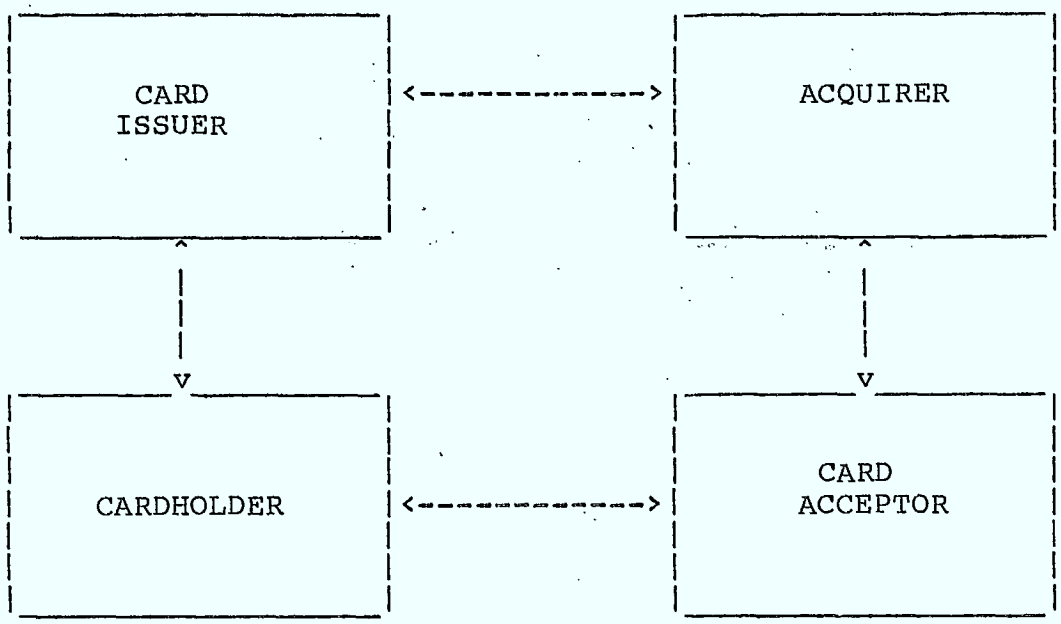


Figure 6.1 Processes Involved in Interactive Authorization Application

6.4 Starting Requirement

The essential information that must be available before the methodology can proceed is a knowledge of which processes in the distributed application require interaction, in the OSI sense, with other processes.

Where an application involves only two processes in separate physical systems, then there is no problem.

Where an application involves more than two processes, it may be necessary to analyze the possible physical and organizational mappings to determine:

1. which processes need to interact via a communications service; if two processes are always co-located in the same physical system, then there is no requirement for a communications protocol between them;

2. how many different protocols are required; if the nature of the interactions between subsets of processes differ considerably, then more than one protocol may be required to satisfy the communication requirements of the application;

3. whether each identified protocol needs to be standardized; if two processes which require a protocol for communication are both located within the same organization or the same management domain, then a standardized protocol may not be necessary.

****
*

One example of this form of analysis is provided in the CCITT X.400 Recommendation on Message Handling Systems.

This Recommendation defines a system model with three types of processes: users, user agents and message transfer agents. It recognizes the possibility of different physical and organizational mappings; an example of a physical and an organizational mapping is provided in Figure 6.2 (a) and (b), respectively. It identifies four types of interactions: 1) among user agents; 2) among message transfer agents; 3) between a user agent and a message transfer agent; and 4) between a user and a user agent.

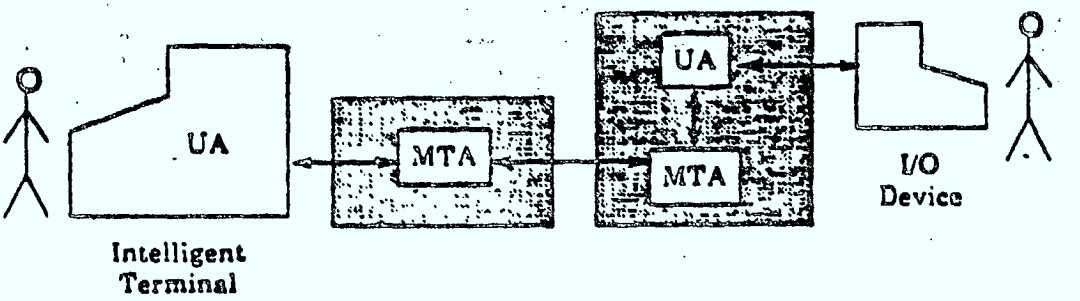


Figure 6.2 (a) Example of Physical Mapping for Message Handling Systems

- - - - denotes organization boundaries

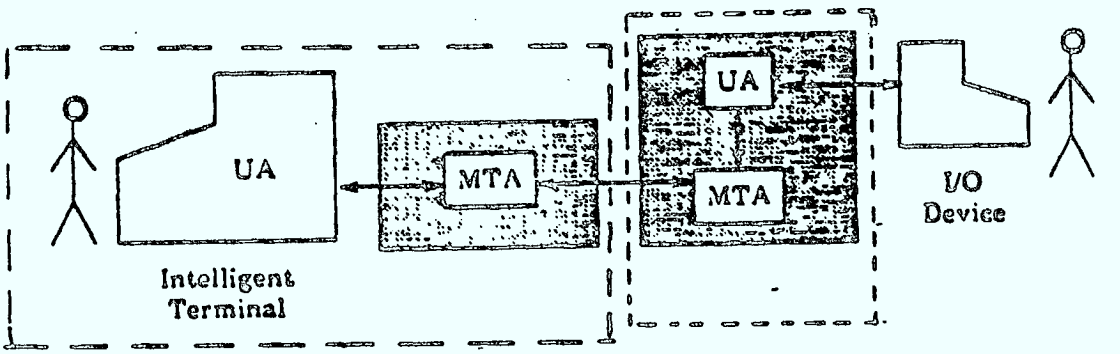


Figure 6.2 (b) Example of Organizational Mapping for Message Handling Systems

In recognition of the different possible physical and organizational mappings, the first three of these types of interactions are made candidates for standardization. As a result, three different protocols (P1, P2 and P3) are developed in companion recommendations.

*
****

****
*
6.4.1 Application to Interactive Authorization

In the case of the IA application, three processes and three types of interactions are involved.

There are two significant physical configurations: one where the Card Acceptor, Acquirer and Card Issuer processes are located on different physical systems, and one where the Acquirer and Card Issuer are co-located. These are shown in Figure 6.3 (a) and (b), respectively.

In the first case, there is a possible requirement for two protocols. However, consideration of organizational mappings reveals that the Card Acceptor and Acquirer are part of the same organization, i.e. they are in the same management domain and hence there is no need for a standard protocol between these two systems, i.e. they need not be open in the OSI sense. There remains the interaction between the Acquirer and the Card Issuer, which does qualify as an OSI protocol.

In the second case, there is a communication requirement only for the interaction between the Card Acceptor and the Acquirer. Again these two systems form part of the same organization and there is no requirement for an OSI protocol.

*
****

6.5 Step 1: Develop OSI View of Application

The first step is to develop a conceptual view of the application that is consistent with OSI architectural principles.

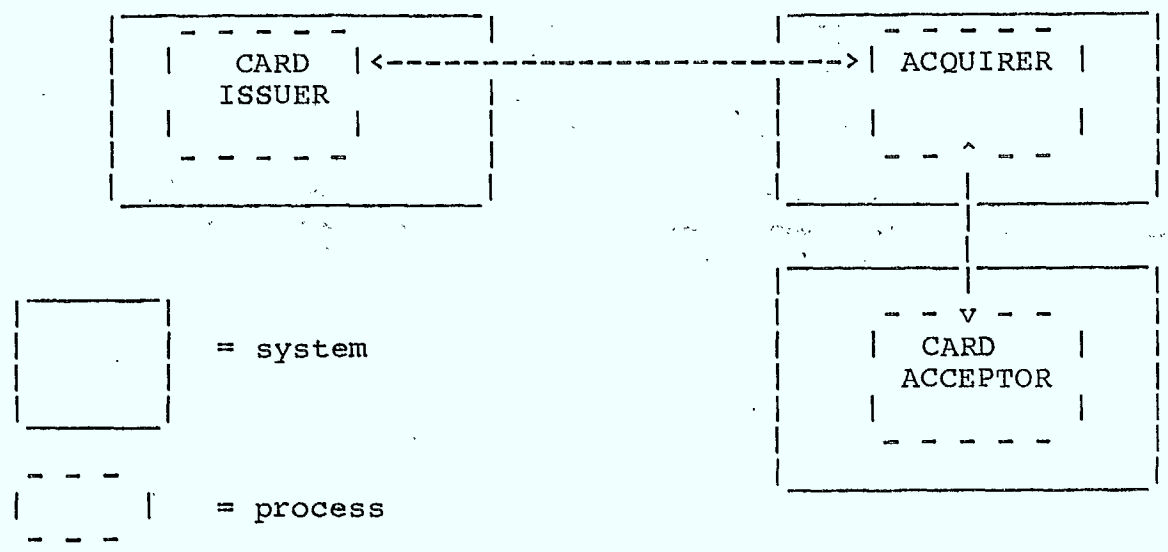


Figure 6.3 (a) Separate Systems for Acquirer, Card Issuer and Card Acceptor

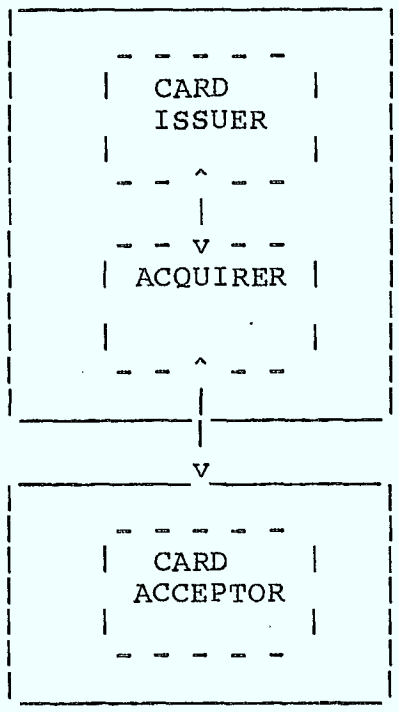


Figure 6.3 (b) Co-location of Acquirer and Card Issuer

This consists of modelling the application in terms of a collection of service users which communicate via a service provider, defining the relationships among the service users and identifying the principal data structures that are meaningful to the application.

A characteristic of this model is that the application entity which initiates associations with one or more other entities is always called the "initiator" while all other application entities are called "responders".

6.5.1 Relationships Among Users

The relationships among the users may be defined in terms of control of services and/or the nature of the processing performed by a particular user.

The types of relationships based on control of services include:

peer-to-peer: a peer-to-peer relationship between two users is one where each user can autonomously initiate associations with the other user.

master-slave: a master-slave relationship between two users is one where one user can initiate a service while the other user can only respond. Typically, once an association has been established, only the initiator can invoke services. In certain cases, a responder may also be allowed to invoke a service, but such cases are restricted normally to the reporting of exceptional conditions such as a local system failure.

The types of relationship based on nature of processing include:

client-server: a client-server relationship between two users is one where one user manages a resource which the other user wishes to have access to. This type of relationship is typically also a master-slave one. A shared file server is an example of a provider which interacts with various users who wish to retrieve (consume) files.

source-sink: a source-sink relationship between two users is one where there is movement of data from the source user to the sink user. The distinction between this relationship and the client-server

one is that the client-server relationship does not necessarily imply movement of data. For example, a request to delete a file is a legitimate client-server interaction but not a legitimate source-sink one. A request to transfer a file to the requestor qualifies as both.

initiation-execution: an initiation-execution relationship between two users is one where one user is capable of executing some operation that another user wishes to have performed. An example of this type of relationship is a remote job entry application.

These various types of relationship are not mutually exclusive. Indeed, all of them may be possible within a single application.

6.5.2 Identification of Principal Conceptual Data Structures

In any distributed application, communication between processes is for the purpose of exchanging meaningful information. For the information to be meaningful, the processes must have a common understanding of the application. This common understanding can be described in terms of a conceptual data structure.

A conceptual data structure is an abstract definition of a data item or of a set of related data items that is meaningful to the communicating parties. It provides a common model for describing the information that is of mutual interest and hides the differences in style and specification that individual systems may use locally to represent such information.

There is no precise rule for determining the nature of this data structure, but the type of relationship between the service users may provide some guidance. For example, if the relationship is a master-slave/producer-consumer one, then the conceptual data structure will typically relate to the nature of the information managed by the producer. If the relationship is an initiation-execution one, then the data structure will likely be concerned with representing the flow of control between the users. For other relationships such as peer-to-peer, then the determination of an appropriate conceptual data structure may not be so obvious and the characteristics of the application itself will be the decisive factor.

Once the major data structures have been identified, it is necessary to describe their internal structure. The emphasis at this stage is on the functional nature of the data structures and not on their precise representation. This can be done in a process of successive refinement. The first step is to identify the type of information that must be present in the conceptual data structure; the next step is to explicitly identify the data elements that satisfy these requirements and state their purpose.

There is no formal specification technique or recommended format for the resulting description.

****
*

This can be illustrated by its application to existing ISO standards.

In the case of the ISO File Transfer, Access and Management (FTAM) standards, the type of relationship can be characterized as a master-slave/client-server one. The principal conceptual data structure is the virtual filestore. The virtual filestore is an abstract representation of a file system. It provides a common model for use in describing files and their attributes and hides the differences in the way real (i.e. physical) systems actually store data and provide access to it. Each system which implements these standards is responsible for developing a mapping between the abstract representation of a file system used by the FTAM protocol and the actual characteristics of the local system.

The principal components of the virtual filestore are files. Files have attributes, some of which describe general properties such as filename and some of which describe the structure of the file, e.g. flat, hierarchical, etc. The complete filestore definition provided in ISO 8571/2 includes a definition of all file attributes.

In the case of the ISO Job Transfer and Manipulation (JTM) standards, the relationship between JTM users is that of initiation-execution with elements of source-sink. The principal conceptual data structure is the work specification. This specification defines in a defined way the work that is to be performed as part of a job. It contains fields, which for example provide

for the identification of the work to be performed, for its authorization, for identifying who is to perform the work, for identifying what reporting information is to be supplied, and for specifying further work to be performed when initial work is completed. A work specification also may contain one or more documents which convey the actual information that is to be processed by a recipient JTM user. For example, the actual Job Control Language that is to be processed by a particular computer system would be contained in a document. Any document type can be transferred between JTM users, as long as there exists an abstract syntax for it.

* *
****

6.5.3 Sub-layering

In some cases, an application cannot be described adequately by a single conceptual data structure or related set of data structures. Multiple levels of data structure (i.e. of abstraction) may be more appropriate.

The concept of sub-layering may apply in such cases. The service provider is decomposed into sub-layers where the upper sub-layer, i.e. the one closer to the end user, acts as a user of the next lower sub-layer. Different conceptual data structures are associated with each sub-layer, but they are hierarchically related in that the data structure at one level is nested within the data structure associated with the next lower sub-layer.

For sub-layering to apply, it is not sufficient that there be a hierarchy of data structures. For example, a JTM work specification is itself a complex data structure containing many data elements, some of which in turn may be complex data structures, e.g. a document. It is necessary that the different data structures be at different levels of abstraction in terms of the communications aspects of the application.

Sub-layering is appropriate when the following criteria are satisfied:

- there are different conceptual data structures which are hierarchically related;

- the data structures are operated upon at different levels of abstraction;

- each sub-layer adds services to those provided by the sub-layer below;

- each sub-layer requires one or more distinct peer protocols.

****
*

An example of sub-layering is the CCITT Message Handling Systems (MHS) set of recommendations. In this case, there are two principal conceptual data structures, each of which is meaningful to a different sub-layer.

The User Agent sub-layer is concerned with an "Interpersonal message" (IP-message) consisting of a heading and a body. The body is the actual information that is to be conveyed to the recipient user while the heading contains information relevant to the processing of the body.

The Message Transfer sub-layer is concerned with a "message" consisting of an envelope and a content. The envelope carries information relevant to the transfer of the message to its destination(s) while the content is the information delivered to the recipient(s). The IP-message and message are related in that the content of the message is the IP-message.

*
****

****
*

6.5.4 Step 1 for Interactive Authorization Example

The OSI view of this application is shown in Figure 6.4 There are only two users, namely the Acquirer and Card Issuer.

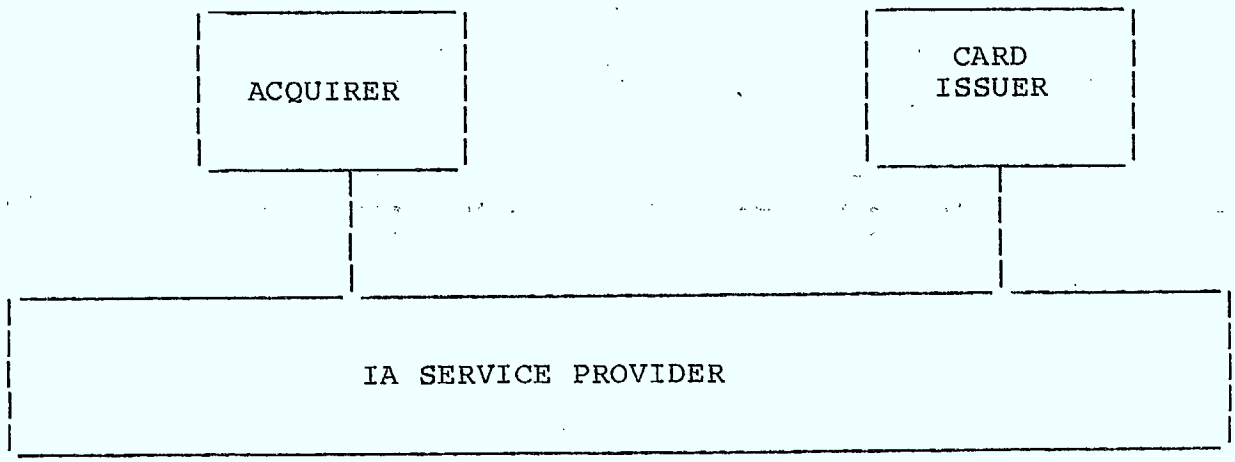


Figure 6.4 OSI View of IA Application

6.5.4.1 Roles of the Users

The Acquirer can be only an initiator and the Card Issuer can be only a responder. There is a master-slave, initiation-execution relationship between these two users in the sense that the Acquirer asks the Card Issuer to perform an authorization function to which request the Card Issuer is expected to respond.

6.5.4.2 Conceptual Data Structure

The principal conceptual data structure for this application is the card authorization record. This data structure represents all the pertinent information for the authorization of a bank-card-related financial transaction. It is closer to a MHS message or JTM work specification in that it is a data structure that is exchanged between systems, unlike FTAM where the virtual filestore remains local to each system.

Note that from an implementation perspective, there is another important data structure maintained by the Card Issuer. This is the customer file, which contains the information on which an authorization request is accepted or rejected. However, this information is not visible to the Acquirer and therefore does not play a role in the definition of standards for this application. How an authorization decision is taken is of no concern to the Acquirer -- from its perspective, the decision could be based on information maintained by a human operator. The Acquirer is interested only in getting a response to its request. On the other hand, if the authorization decision was to be made by the Acquirer based on information retrieved from the customer file, then this file would become an important conceptual data structure for this application.

The card authorization record is required to contain the following information:

- identification of the transaction: this uniquely identifies a transaction
- account information: identifies the customer account
- identification of the transaction participants: indicates who the Card Acceptor, Acquirer and Card Issuer are

- currency information: identifies the currency of the transaction

- transaction amount: the value of the transaction being performed

From this set of requirements can be derived the data elements identified in ISO 8583. Note that the data elements identified in 8583 which relate to the transmission of a message, such as "transmission date and time", are not present in this data structure. Only those data elements which are pertinent to the data processing aspects of the application are included.

There is no requirement for sub-layering.

* *
****

6.6 Step 2: Identify Service Elements

A conceptual data structure in itself does not provide a complete understanding of an application. It is necessary also to specify the operations that are relevant to that conceptual data structure.

For example, in the case of a file transfer and manipulation application, it is necessary to specify the operations that are possible on the virtual filestore.

The interaction between application entities regarding these operations is modelled in terms of invocation of services provided by application service elements. Therefore this step of the methodology is concerned with identification of the service elements relevant to the application.

Temporary Note: Note the distinction in the above paragraph between service elements and services. This is consistent with the discussion in clause 5.4.2.1.

The emphasis at this step is on informal description. A formal specification will follow in a subsequent step.

6.6.1 Criteria for Selection of Service Elements

There is no precise rule for this. The definition of the term "service element" is sufficiently broad that any OSI capability of a service provider can be labelled a service element.

The following selection process is recommended.

1. if there is more than one conceptual data structure, and sub-layering applies, then a separate set of service elements is identified for each sub-layer. Separate service definitions and protocol specifications must be produced for each sub-layer.

2. allocate a service element for each operation that directly involves a conceptual data structure. This is the single most important selection criterion; it requires a thorough understanding of the application. For example, for a file transfer application, a service element would be allocated for selecting and deselecting a file, for opening and closing a selected file, reading and writing, etc. These service elements are specific to the application and hence are categorized as SASE.

3. determine whether the application is connection-oriented, connectionless or store-and-forward in nature. This decision is related to the quality of service requirements of the application.

   Does the application require reliable transfer of large amounts of data? Does it require negotiation of the characteristics of the interaction between communicating partners? Or does it require an ongoing dialogue between the communicating partners? If so, then a connection-oriented mode of communication is appropriate.

   Does the application have a requirement for low data transfer delay and for the transfer of small amounts of data? Is reliability of data transfer a secondary requirement? Is there a need to send the same information to multiple destinations? Is negotiation of communications characteristics unnecessary? If so, then a connectionless mode of communication may be appropriate.

Does the application have a requirement for reliable transfer of small to medium amounts of data to one or more destinations which may or may not be always available? Is transfer delay of secondary importance? If so, then a store-and-forward mode of communication may be appropriate.

If the mode of communication is to be connection-oriented, then service elements are required for initializing and terminating the application context. Three such service elements are usually needed: one to initialize the application context, one to terminate it gracefully and one to abort it unconditionally. These service elements may provide for the exchange of information concerning variable aspects of the application context, e.g. selection of functional units or provision of accounting information upon termination of the context.

4. allocate a service element for provider functions that are not directly related to manipulation of the conceptual data structure(s) but do satisfy a communication requirement of the application. Such communication requirements are derived from an analysis of the conceptual model of the application developed in step 1 and an intuitive appreciation of the underlying functionality needed to support the service elements identified in the first step of the selection process.

   Included in this category are the mandatory CASE service elements (e.g. A-ASSOCIATE, A-RELEASE, etc.) and those optional CASE service elements which are appropriate, e.g. Commitment Control and Recovery service elements. Also included here are other SASEs which are meaningful to the application, e.g. a file transfer SASE to provide reliable file transfer in support of a distributed banking application. All supporting service elements identified in this step belong to the lowest sub-layer of the application entity. Clause 7 provides a more complete discussion of supporting services.

4. exclude any functionality that is strictly local to an end system.

The result of this process is a complete factorization of the application entity functionality. The internal structure of the application entity is now understood and the number and nature of different sets of SASEs and corresponding protocols is known.

Each identifiable SASE set corresponds to a different application context.

****
*

6.6.2 Step 2 for Interactive Authorization Example

6.6.2.1 SASE

The conceptual data structure for this application is the card authorization record. The following service elements correspond to operations relevant to this data structure:

Authorization: determines whether a financial transaction is authorized to proceed.

Authorization Repeat: repeats an outstanding authorization request for which a response has not been received.

Authorization Completion: indicates that authorization actions specified by Card Issuer have been completed.

Authorization Completion Acknowledgement: acknowledges authorization completion.

Authorization Completion Repeat: repeats an authorization completion for which an acknowldegement has not been received.

Acquirer Reversal: reverses an earlier authorization.

Acquirer Reversal Repeat: repeats an Acquirer reversal request for which a response has not been received.

Card Issuer Reversal: reverses an earlier authorization.

Card Issuer Reversal Repeat: repeats a Card Issuer reversal request for which a response has not been received.

6.6.2.2 Style of Communication

This application has a requirement for fast response time. The number of interactions between an Acquirer and Card Issuer per financial transaction is typically two. These characteristics suggest the use of either connection-oriented or connectionless supporting services.

A connectionless service has the advantage of potentially smaller communications delay because of less overhead. It has the disadvantages that its reliability is totally dependent on low-level network characteristics as presently there are no higher-level error recovery capabilities specified in existing connectionless protocols, and that there is no opportunity for negotation of association characteristics as these are fixed by a priori agreement.

A connection-oriented service may have increased communications delay, but this is in large measure due to error recovery capabilities which are important for this application. It also provides the capability for dynamic negotiation of association characteristics, which is advantageous if requirements change over time. Potentially, the most serious drawback is the connection establishment delay which can be significant if a Network connection must be established for each authorization request. Delay can be reduced somewhat by maintaining a Network connection open indefinitely and establishing separate application associations and Presentation, Session and Transport connections for each authorization transaction. A third alternative is to establish an association only once and keeping it open indefinitely. This eliminates connection establishment delay on a per-transaction basis at the cost of permanently allocating communications resources such as buffers.

A connection-oriented approach is used in this example. Hence, three additional specific application service elements are required:

Initialization: initialize the interactive authorization context.

Termination: release the context gracefully so that no data is lost.

Abort: terminate the context abruptly, with possible loss of data.

The CASE kernel set of service elements is also required. This set of service elements provide for association establishment and release.

As only two parties are involved in this application and only small quantities of data are being transferred, no requirements exist for Commitment, Concurrency and Recovery or reliable document transfer service elements.

* *
****

6.7 Step 3: Prepare Service Definition

The previous step defined the functionality of the service provider in terms of service elements. The next step is to specify in an abstract way how this functionality is made available to service users.

6.7.1 Service Conventions

The method described here is based on the service conventions contained in TR 8509. It applies to the definition of a service involving one service provider and two service users.

Conceptually, the functionality of the service provider is accessed through the invocation of services. A service is described in terms of a set of interactions between service users and the service provider. Each abstract, implementation-independent unit of interaction between a single service user and the service provider is called a service primitive.

There are four types of service primitive:

1. request: a service primitive issued by a service user to invoke some procedure.

2. indication: a service primitive issued by a service provider either:

   a) to invoke some procedure; or

   b) to indicate that a procedure has been invoked by the peer service user.

3. response: a service primitive issued by a service user to complete some procedure previously invoked by an indication to that same service user.

4. confirm: a service primitive issued by a service provider to a service user to complete some procedure previously invoked by a request of the service user.

The occurrence of a service primitive is a logically instantaneous and indivisible event. It cannot be interrupted by another event.

One or more parameters may be associated with a service primitive and each of these parameters has a defined range of values. Parameter values associated with a service primitive are passed in the direction of the service primitive.

Services are described in terms of groups of service primitives. The occurrence of a group of service primitives is not a logically instantaneous event. The intervals between the constituent service primitives may be non-disruptively interspersed with other service primitives, according to sequencing rules defined for the service.

The principal types of service are:

1. unconfirmed service: a service in which a request issued by one service user leads only to an indication issued to the other service user.

2. confirmed service: a service in which a request issued by one service user leads to an indication to the other, which provokes the service user into issuing a response, leading to a confirm being issued to the originating service user. An alternative form of confirmed service is one where there are no indication or response primitives, just a request and a confirm.

3. provider-initiated service: a service which is generated by the service provider. It consists of indication service primitives issued to each service user.

6.7.2 Elements of a Service Definition

The following information should be present in a service definition standard:

- Service model and service elements: this information should be present here if not specified in a separate document

- Identification of services: this includes a list of services, a brief description of their purpose, their mapping onto service elements and their classification according to service type.

- Groupings of services: this specifies meaningful collections of services in terms of service subsets, service classes or functional units.

- Definition of primitives: this includes a definition of all parameters and their characteristics.

- Sequences of primitives: this describes the constraints on the sequencing of service primitives.

Each of these elements is discussed further below.

6.7.2.1 Service Model and Service Elements

See Clauses 6.5 and 6.6.

6.7.2.2 Identification of Services

Three items of information are required: a list of service names, their corresponding service type and their corresponding mapping onto service elements.

A tabular format is appropriate for presenting this information.

Service names should adhere to the following convention:

- use capital letters;
- the first one or two letters should represent the application,
- the application identifier should be separated from the service identifier by a hyphen.
- a descriptive name should be given to the service, e.g. "OPEN" to open a file. Thus the complete name for a file transfer application file open service would be "F-OPEN".

A service primitive is identified by appending the primitive name to the service name, e.g. F-OPEN request.

The classification into service type will indicate whether the service is confirmed, unconfirmed or provider-initiated.

There are two ways in which a service element can be mapped onto a service:

1. direct mapping to service: This corresponds to a one-to-one mapping between a service element and a service. This is the usual mapping. It is appropriate when a service element represents a distinct capability of the service provider that can be invoked separately from other service elements.

2. mapping to primitive parameter: In this case, a service element is mapped to a parameter of a service primitive. It is appropriate when a service element does not represent a separately-invocable capability, but rather is one of a related group of capabilities which can be controlled by a single service invocation. An example of this form of mapping is found in the CCITT MHS specifications.

6.7.2.3 Groupings of Services

There are three ways in which services can be grouped: functional units, service classes and service subsets.

Functional units represent small meaningful groupings of services. The criterion for such groupings is that a particular member of a group would not be useful if the other members were not also available.

Service classes represent combinations of functional units. There is no precise criterion for selecting a class; it may be done on the basis of gradation of functionality, as in JTM where the basic class is a subset of the full class, or it may be done on the basis of differentiation of functionality, as in FTAM where three different classes are defined to address different aspects of filestore manipulation. A class may optionally include certain functional units and require others. The use of optional functional units must then be negotiated during association establishment.

A subset is similar to a class, but subsets are not standardized and therefore cannot be used during association establishment for negotiating service capabilities. They may be defined as a convenient mnemonic for typical combinations of functional units.

6.7.2.4 Definition of Primitives

For each parameter of each service primitive of each service, its definition, its usage and its range of values must be provided. Also, for each service, the service user which may issue the request primitive must be identified.

The definition of each parameter states its meaning. Only one definition need be provided for a parameter which appears more than once in the specification.

The usage of each parameter states whether the parameter is mandatory, optional or conditional. A mandatory parameter must always be present when the service primitive is invoked. An optional parameter may be present if appropriate. The interpretation of the absence of a parameter is parameter-dependent and must be explained as part of the specification. A conditional parameter is one whose usage is dependent on the presence or value of other parameters. This dependency must be explained as part of the specification.

3. provider-initiated service: a service which is generated by the service provider. It consists of indication service primitives issued to each service user.

6.7.2 Elements of a Service Definition

The following information should be present in a service definition standard:

- Service model and service elements: this information should be present here if not specified in a separate document

- Identification of services: this includes a list of services, a brief description of their purpose, their mapping onto service elements and their classification according to service type.

- Groupings of services: this specifies meaningful collections of services in terms of service subsets, service classes or functional units.

- Definition of primitives: this includes a definition of all parameters and their characteristics.

- Sequences of primitives: this describes the constraints on the sequencing of service primitives.

Each of these elements is discussed further below.

6.7.2.1 Service Model and Service Elements

See Clauses 6.5 and 6.6.

6.7.2.2 Identification of Services

Three items of information are required: a list of service names, their corresponding service type and their corresponding mapping onto service elements.

A tabular format is appropriate for presenting this information.

Service names should adhere to the following convention:

- use capital letters;
- the first one or two letters should represent the application,
- the application identifier should be separated from the service identifier by a hyphen.
- a descriptive name should be given to the service, e.g. "OPEN" to open a file. Thus the complete name for a file transfer application file open service would be "F-OPEN".

A service primitive is identified by appending the primitive name to the service name, e.g. F-OPEN request.

The classification into service type will indicate whether the service is confirmed, unconfirmed or provider-initiated.

There are two ways in which a service element can be mapped onto a service:

1. direct mapping to service: This corresponds to a one-to-one mapping between a service element and a service. This is the usual mapping. It is appropriate when a service element represents a distinct capability of the service provider that can be invoked separately from other service elements.

2. mapping to primitive parameter: In this case, a service element is mapped to a parameter of a service primitive. It is appropriate when a service element does not represent a separately-invocable capability, but rather is one of a related group of capabilities which can be controlled by a single service invocation. An example of this form of mapping is found in the CCITT MHS specifications.

6.7.2.3 Groupings of Services

There are three ways in which services can be grouped: functional units, service classes and service subsets.

Functional units represent small meaningful groupings of services. The criterion for such groupings is that a particular member of a group would not be useful if the other members were not also available.

Service classes represent combinations of functional units. There is no precise criterion for selecting a class; it may be done on the basis of gradation of functionality, as in JTM where the basic class is a subset of the full class, or it may be done on the basis of differentiation of functionality, as in FTAM where three different classes are defined to address different aspects of filestore manipulation. A class may optionally include certain functional units and require others. The use of optional functional units must then be negotiated during association establishment.

A subset is similar to a class, but subsets are not standardized and therefore cannot be used during association establishment for negotiating service capabilities. They may be defined as a convenient mnemonic for typical combinations of functional units.

6.7.2.4 Definition of Primitives

For each parameter of each service primitive of each service, its definition, its usage and its range of values must be provided. Also, for each service, the service user which may issue the request primitive must be identified.

The definition of each parameter states its meaning. Only one definition need be provided for a parameter which appears more than once in the specification.

The usage of each parameter states whether the parameter is mandatory, optional or conditional. A mandatory parameter must always be present when the service primitive is invoked. An optional parameter may be present if appropriate. The interpretation of the absence of a parameter is parameter-dependent and must be explained as part of the specification. A conditional parameter is one whose usage is dependent on the presence or value of other parameters. This dependency must be explained as part of the specification.

In certain cases, a parameter of a service primitive is constrained to be identical to that of a previous primitive of the same service. For example, a filename associated with a "file open" indication must be identical to the filename provided with the "file open" request. Such constraints must be stated.

A tabular format is helpful to illustrate usage. A brief example is provided below. In this example, a parameter of an indication or response primitive that is constrained to take on the same value as its predecessor primitive is indicated with an "(=)". A blank space under a service primitive indicates that the corresponding parameter is not applicable to that primitive.

Parameter	F-DELETE request	F-DELETE indication	F-DELETE response	F-DELETE confirm
Diagnostic			Mandatory	Mandatory
Delete Password	Optional	Optional (=)		
Charging			Optional	Optional

The range of values for each parameter must be specified, as well any default value if applicable. A formal definition of parameter syntax in terms of primitive types, such as integers, strings, boolean, etc., is desirable. Use of the ASN.1 syntax notation is recommended (see Clause 8).

Some services can be invoked by either service user, others by only one. This information must be supplied.

6.7.2.5 Sequences of Primitives

The interrelationships and valid sequences of primitives must be specified. There are two aspects to this information:

1. the "local rules" which determine the possible sequences of primitives involving one service user and the service provider. These local rules may be different for the initiator and responder service

users. These rules are often represented by a state transition diagram which illustrates the allowed order of service primitives. When the service being specified is complex, it may be appropriate to represent only normal interactions via a state transition diagram, and to use a tabular format to represent error conditions.

2. the "end-to-end" properties of the service which relate the service primitives interactions of the service provider with each of the service users.

Local rules and end-to-end properties, together, may be specified by example, using the notation of time sequence diagrams.

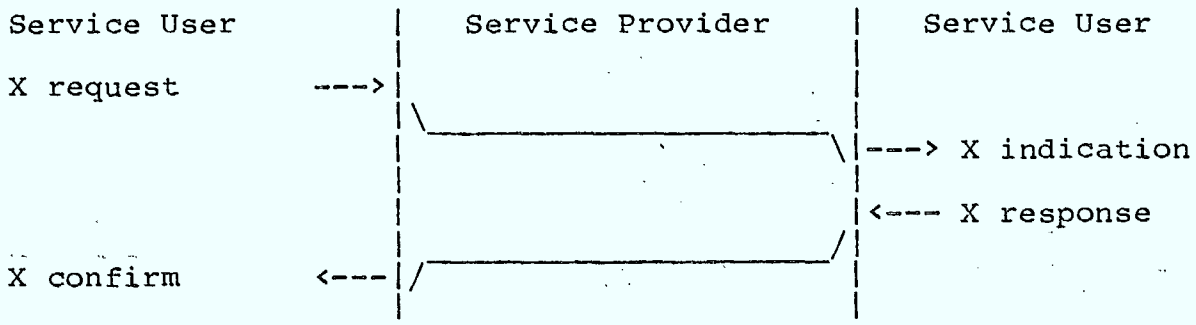
Each time sequence diagram is partitioned by two vertical lines into three fields. The central field represents the service provider and the two side fields represent the two service users.

Sequences of events are positioned along lines representing the passage of time, increasing downwards.

Arrows, placed in the areas representing the service user, indicate the direction of propagation of primitives, i.e. to or from the service provider.

Necessary sequence relations between peer service users are emphasized by an arrow between the time lines. In the absence of this arrow, or when there is a tilde present instead, there is no specific sequence between points in time on the two lines. An example of this is a situation where the provider issues a confirm primitive before the responding service user has issued a response primitive.

The following diagram is a time sequence diagram for a confirmed service.



****
*
*

6.7.3 Application of Step 3 to Interactive Authorization

The service model and list of service elements have already been provided in steps 1 and 2. This example will focus on the identification of services, their groupings, the definition of service primitives and primitive sequences.

6.7.3.1 List of Services

For this application, each service element corresponds to a separate service, so the mapping is one-to-one. The following table lists the services and their primitive group type. The initials "IA" are used to represent the interactive authorization application.

Note that the terminology of ISO 8583 for message types has been modified in this table to avoid conflict with the accepted terminology for OSI service primitives.

Table 6.1 Services for Interactive Authorization

Service Name	Primitive Group Type
IA-INITIALIZE	CONFIRMED
IA-TERMINATE	CONFIRMED
IA-ABORT	UNCONFIRMED
IA-AUTHORIZE	CONFIRMED
IA-AUTHORIZE-REPEAT	CONFIRMED
IA-COMPLETE	UNCONFIRMED
IA-COMPLETE-REPEAT	CONFIRMED
IA-COMPLETE-ACK	UNCONFIRMED
IA-ACQUIRER-REVERSAL	CONFIRMED
IA-ACQUIRER-REVERSAL-REPEAT	CONFIRMED
IA-CARD-ISSUER-REVERSAL	CONFIRMED
IA-CARD-ISSUER-REVERSAL-REPEAT	CONFIRMED

Note that this choice of services places the responsibility of initiating request repeats with the service user. An alternative mapping could have been selected where repeats were provider-initiated, perhaps under the control of a parameter of an initial request.

6.8 Service Groupings

The following functional units may be defined:

kernel functional unit: consists of IA-INITIALIZE, IA-TERMINATE, IA-ABORT, IA-AUTHORIZE, IA-ACQUIRER-REVERSAL, IA-CARD-ISSUER-REVERSAL;

completion functional unit: consists of IA-COMPLETE;

ack functional unit: consists of IA-COMPLETE-ACK;

repeat functional unit: consists of IA-AUTHORIZE-REPEAT, IA-COMPLETE-REPEAT, IA-ACQUIRER-REVERSAL-REPEAT, IA-CARD-ISSUER-REVERSAL-REPEAT;

The use of a particular functional unit is negotiated at time of association establishment. Once a functional unit has been selected, its use becomes mandatory, i.e. if the IA-COMPLETE functional unit is selected, then every an IA-COMPLETE request must be issued whenever an IA-AUTHORIZE confirm is received.

The only restrictions on functional unit combinations are that the ack functional unit can be selected only in conjunction with the completion functional unit and that the kernel functional unit must always be selected.

There is no requirement to identify service classes or subsets.

6.8.Ø.1 Definition of Primitives

This section can include the definition of parameters provided in ISO 8583 Section 4.3, as well as tables 1 and 2.

Section 4.3 of ISO 8583 would be modified to remove data elements that are not appropriate to interactive authorization and to remove references to bit map position.

Table 1 of ISO 8583 would require modification to replace message type identifiers with service primitives and to delete the bit map data elements. These data elements are inappropriate for a service definition as they are concerned with encoding of protocol data units. Parameters that are not appropriate to interactive authorization would also be removed.

Table 2 of ISO 8583 would be modified to remove conditions that do not apply to parameters remaining in the modified table 1.

6.8.Ø.2 Sequences of Primitives

The valid sequences of primitives are indicated by a combination of state transition and time sequence diagrams.

Figures 6.5 and 6.6 show the state transitions possible for context management at each of the user-service-provider interfaces.

Figures 6.7 and 6.8 show the state transitions possible for authorizations at each of the user-service-provider interfaces. Note that in these figures, there are four possible states from which the interface can return to idle. Three of these, namely AUTHORIZED, AUTHORIZE COMPLETION and AUTHORIZE COMPLETION ACK, are states

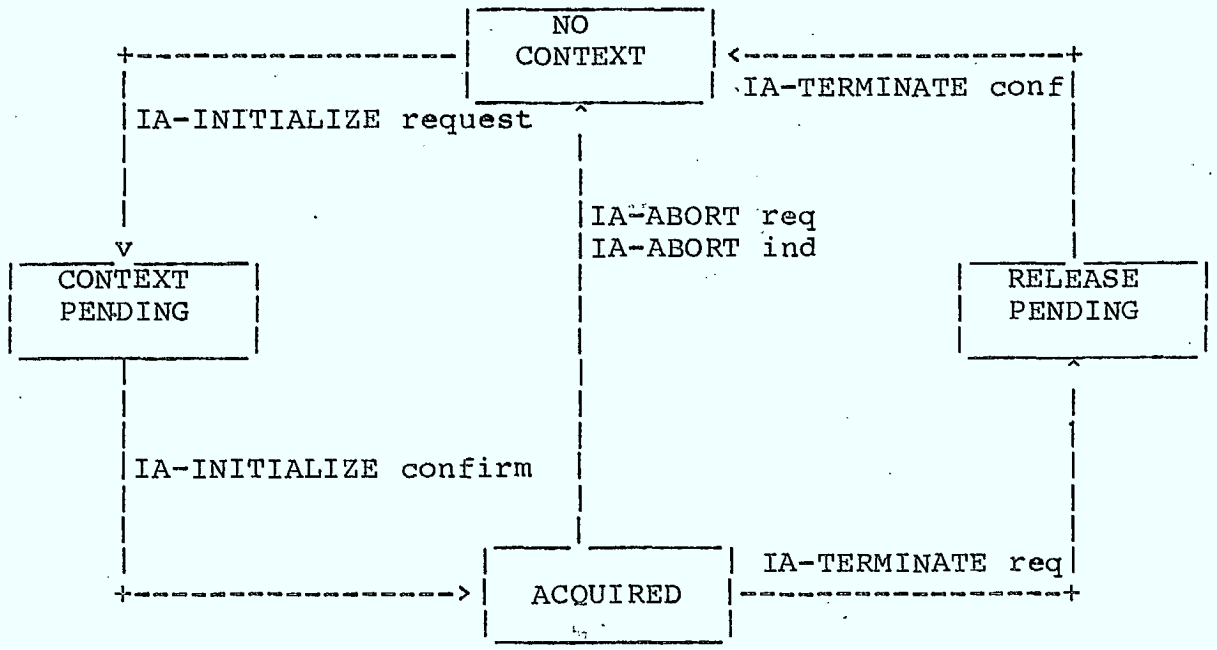


Figure 6.5 Context Management State Transitions at the Interface Between Acquirer and Service Provider

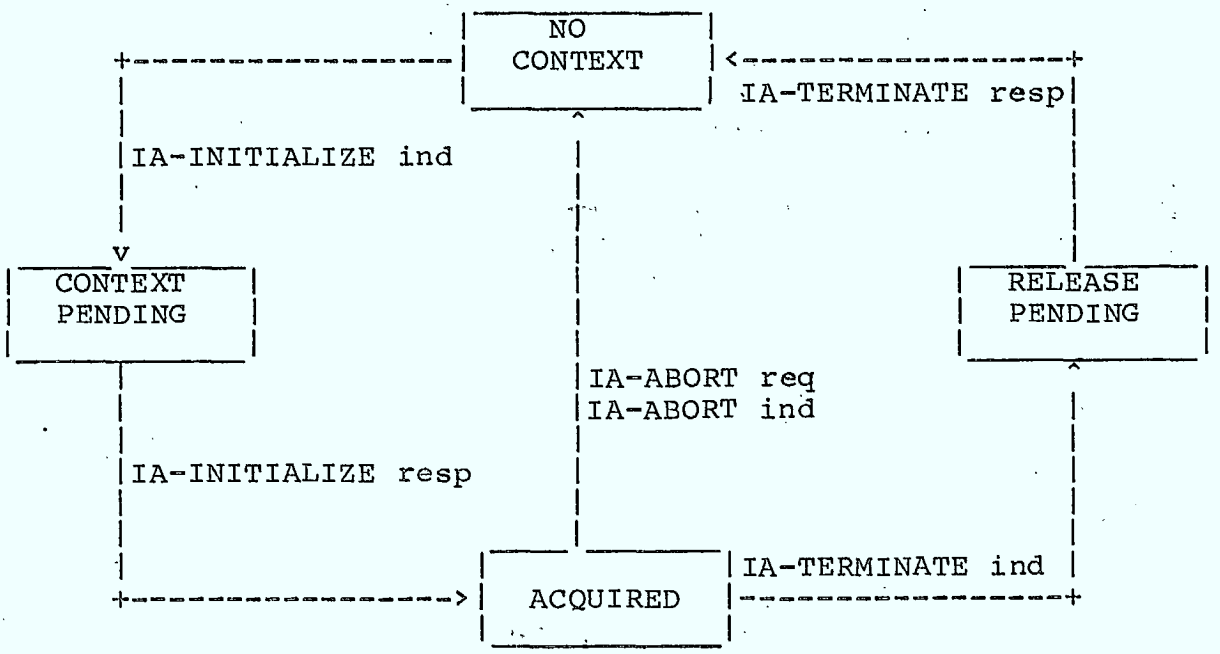


Figure 6.6 Context Management State Transitions at the Interface Between Card Issuer and Service Provider

where no more authorization processing may be performed, depending on the selection of functional units. For example, the AUTHORIZED state represents completion of processing when the completion and completion ack functional units are absent. From each of these states, a reversal service may be invoked. Once authorization processing has completed for a particular financial transaction, there is a finite time in which a reversal can be invoked. After this interval, which is system dependent, no further activity for a particular transaction can be initiated.

Figures 6.9 and 6.1Ø show the state transitions possible for reversals at each of the user-service-provider interfaces. In these figures, note that a reversal can be invoked while the interface is in any of three states; if the reversal fails, then the interface returns to the state before the a transaction can be undertaken and the service provider enters an idle state relative to that transaction.

Figures 6.5 and 6.6 apply to a single association. Figures 6.7 - 6.1Ø apply to a single transaction; these transitions are possible only when the context state is "ACQUIRED". Many transactions may occur during an association and may overlap.

Missing from these diagrams are transitions associated with a repeat service invoked after completion of the service being repeated.

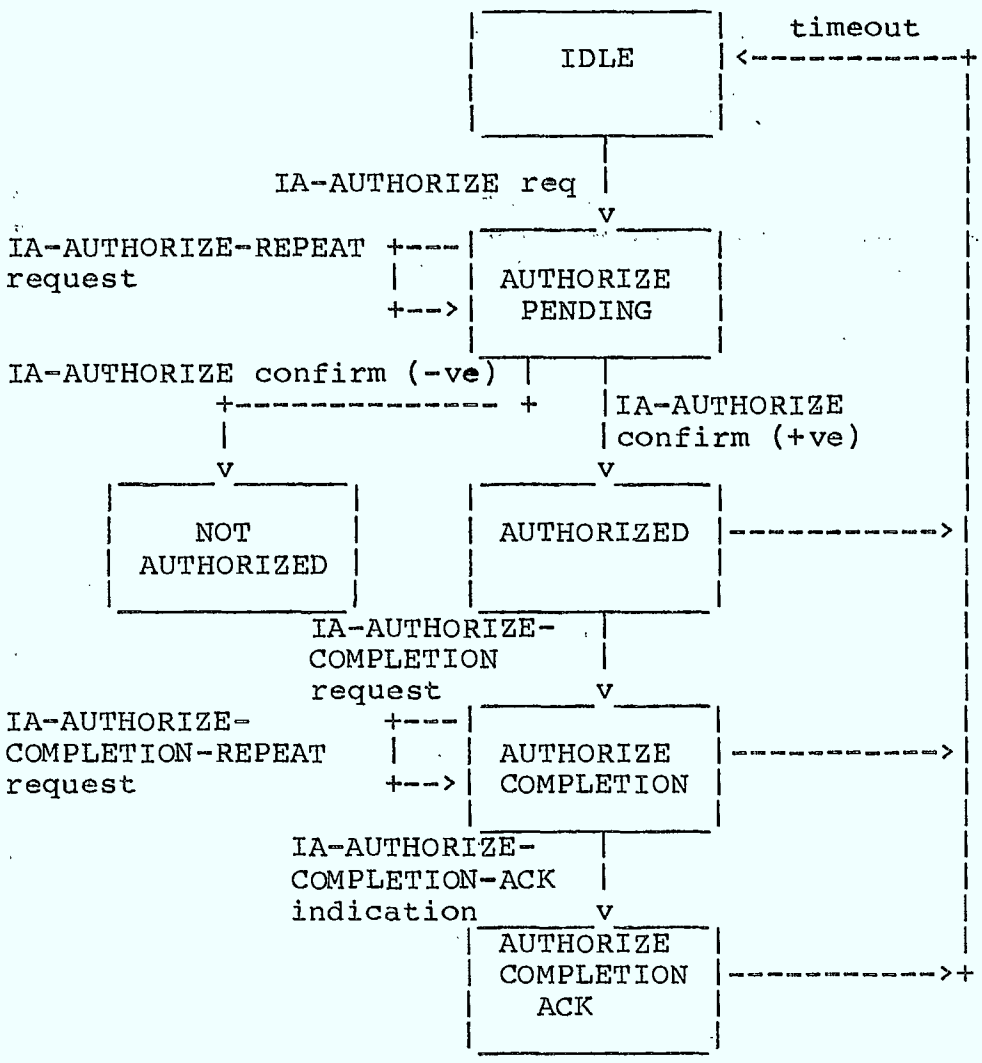


Figure 6.7 Authorization State Transitions at the Interface Between Acquirer and Service Provider

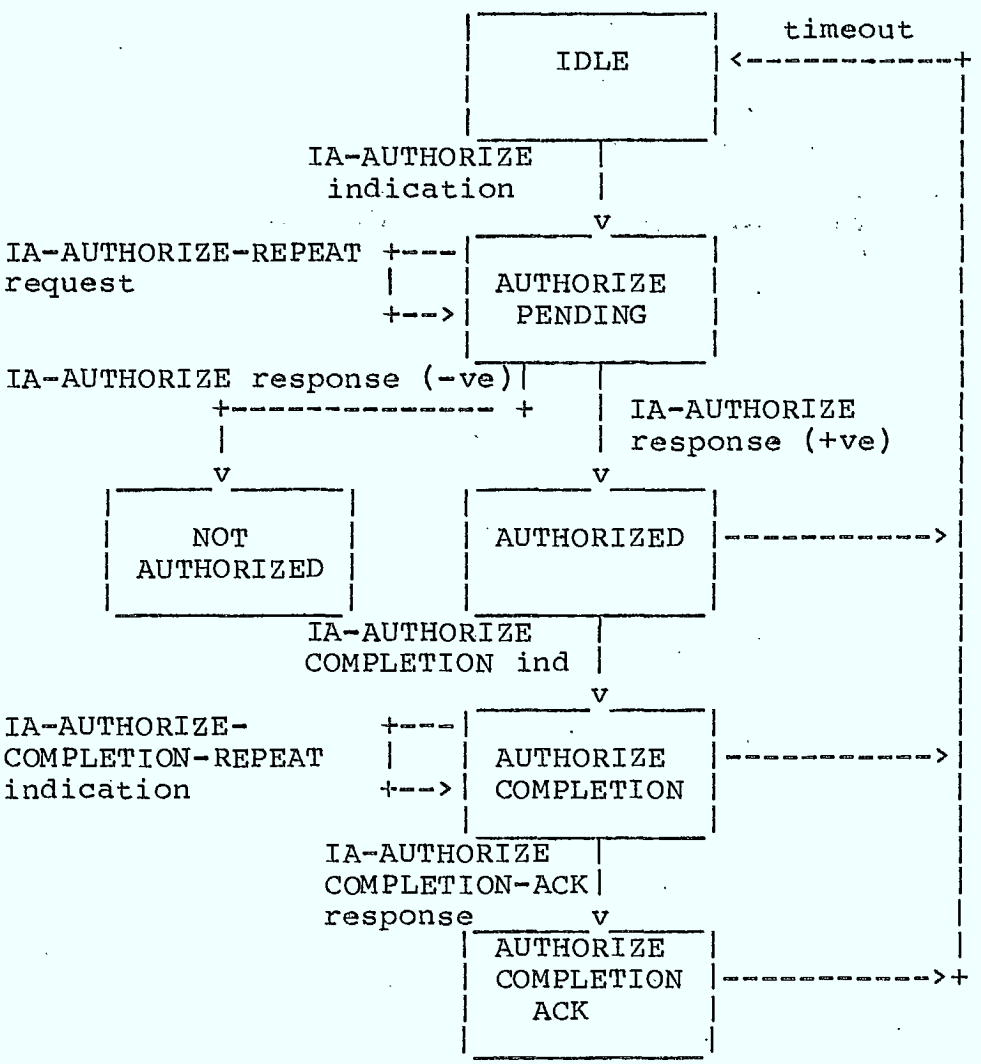
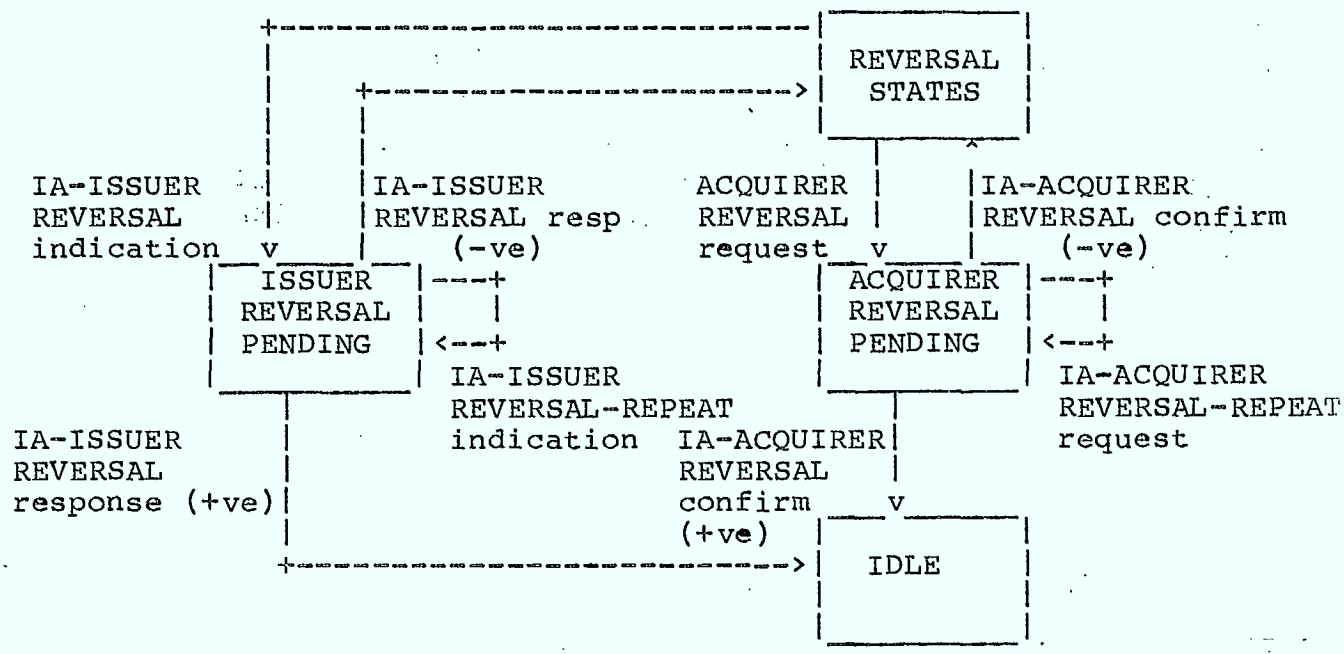
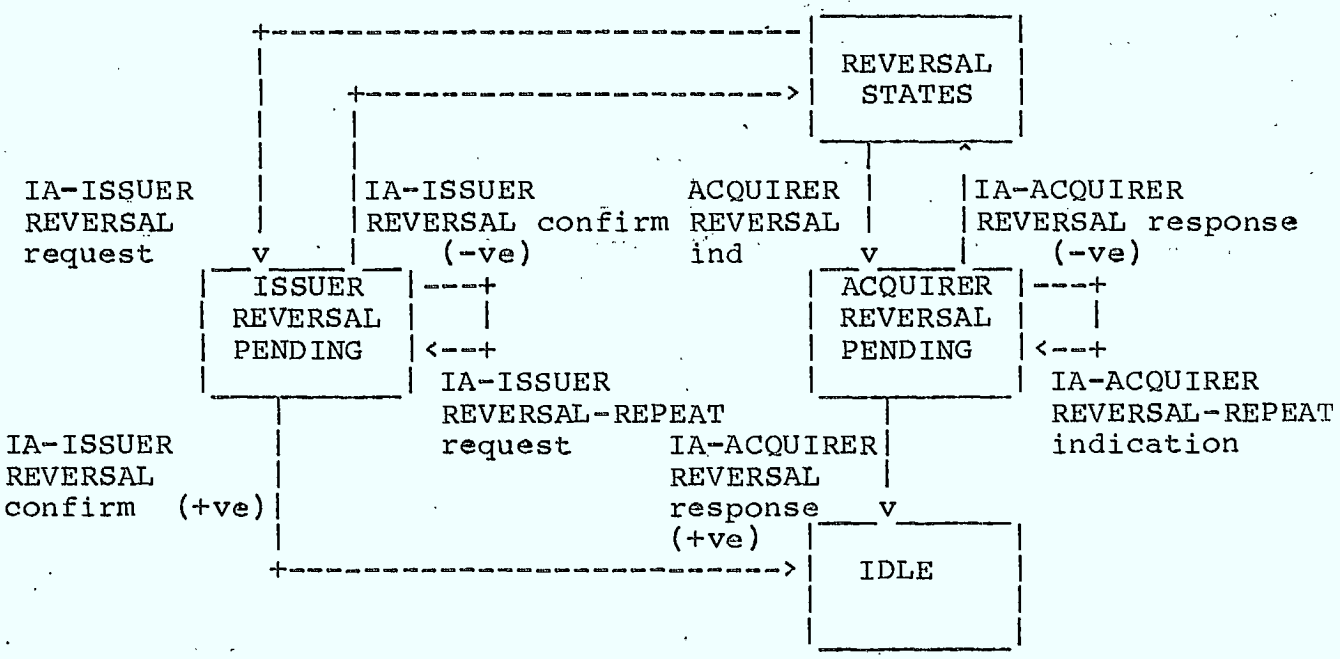


Figure 6.8 Authorization State Transitions at Interface Between Card Issuer and Service Provider



N.B. REVERSAL STATES is one of AUTHORIZED, AUTHORIZE COMPLETION or AUTHORIZE COMPLETION ACK

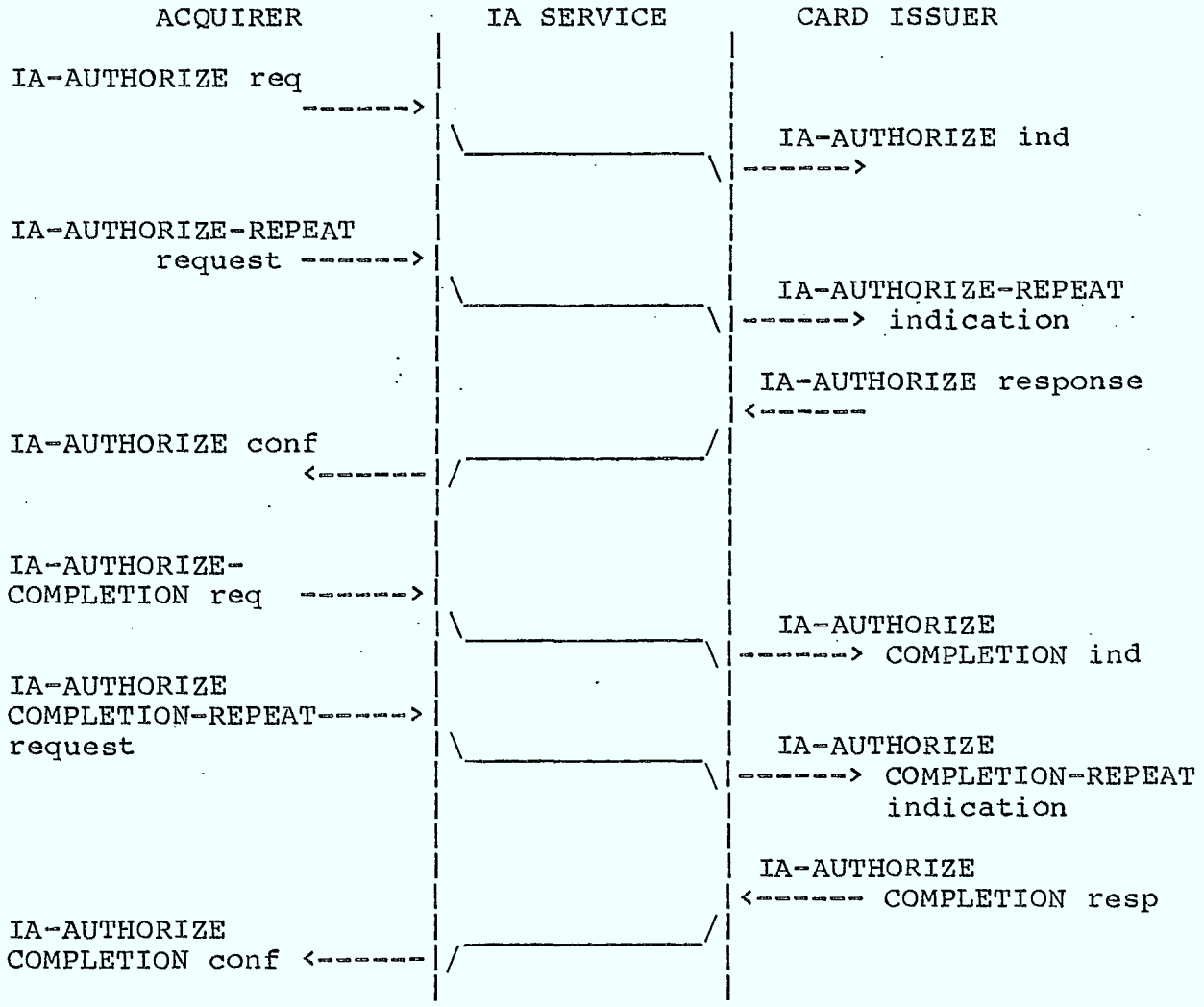
Figure 6.9 Reversal State Transitions at Interface Between Acquirer and Service Provider



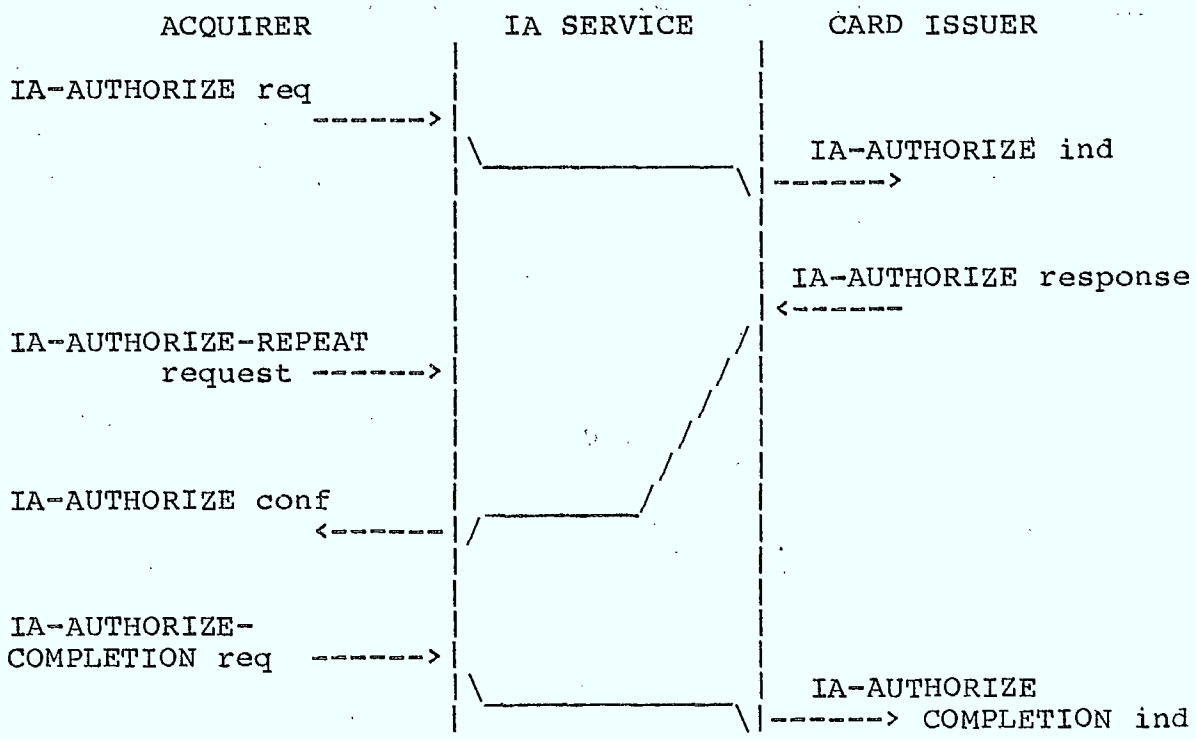
N.B. REVERSAL STATES is one of AUTHORIZED, AUTHORIZE COMPLETION or AUTHORIZE COMPLETION ACK

Figure 6.10 Reversal State Transitions at Interface Between Card Issuer and Service Provider

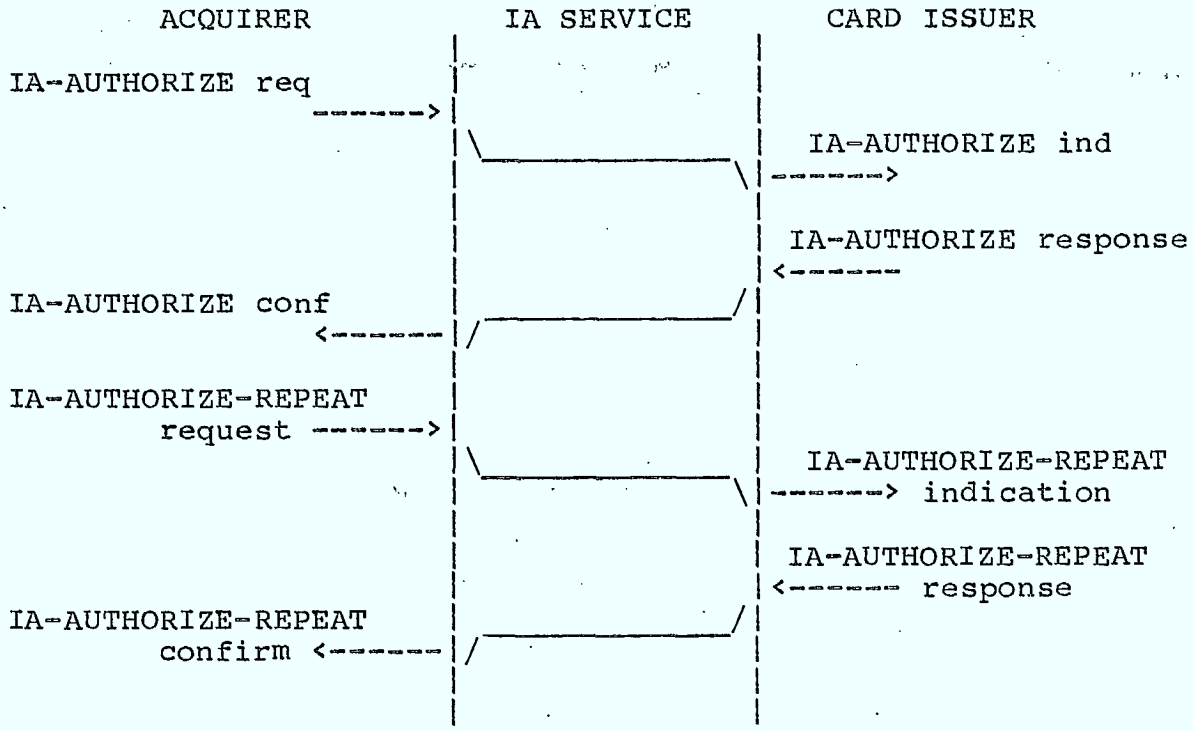
The following time sequence diagram shows the sequence of events associated with an authorization with completion acknowledgement where the IA-AUTHORIZE response primitive is issued after receipt of an IA-AUTHORIZE-REPEAT primitive. Note that in this case the IA-AUTHORIZE response and confirm primitive serve to complete both the IA-AUTHORIZE and IA-AUTHORIZE-REPEAT services.



The following time sequence diagram shows a sequence with authorization and authorization completion without acknowledgement, where a repeat request is issued after an IA-AUTHORIZE response. In this case, no repeat indication is issued to the Card Issuer and the IA-AUTHORIZE confirm completes both outstanding services.



The following time sequence diagram shows an authorization where a repeat is requested after reception of the authorization confirm. In this case, the full repeat service primitive group is involved.



The following time sequence diagram shows Acquirer reversal with repeat.

```
        ACQUIRER              IA SERVICE          CARD ISSUER
                         |                    |
IA-ACQUIRER-REVERSAL     |                    |
        request -------->|                    |
                         |\                   |  IA-ACQUIRER-REVERSAL
                         | \_________________ \|-------> indication
                         |                    |
IA-ACQUIRER-REVERSAL     |                    |
REPEAT  request -------->|                    |
                         |\                   |  IA-ACQUIRER-REVERSAL
                         | \_________________ \|-------> REPEAT indication
                         |                    |
                         |                    | IA-ACQUIRER-REVERSAL
                         |                    |<------- response
IA-ACQUIRER-REVERSAL     |   _________________/|
        confirm <--------|/                   |
                         |                    |
                         |                    |
```

The following time sequence diagram shows Card Issuer reversal with repeat

```
        ACQUIRER              IA SERVICE          CARD ISSUER
                         |                    |
                         |                    | IA-CARD-ISSUER-REVERSAL
                         |                    |<------- request
IA-CARD-ISSUER-REVERSAL  |   _________________/|
     indication <--------|/                   |
                         |                    |
                         |                    | IA-CARD-ISSUER-REVERSAL
                         |                    |<------- REPEAT request
IA-CARD-ISSUER-REVERSAL  |   _________________/|
REPEAT ind      <--------|/                   |
                         |                    |
                         |                    |
IA-CARD-ISSUER-REVERSAL  |                    |
   response ------------>|                    |
                         |\                   |  IA-CARD-ISSUER-REVERSAL
                         | \_________________ \|-------> confirm
                         |                    |
```

*   *
****

6.9 Step 4: Prepare Protocol Specification

This step specifies a protocol to support the service defined in the previous step. An application protocol is a set of rules and formats which determines the communication behaviour of application entities in support of application functions. The specification method described here is based on a particular model which is applicable to all OSI protocols.

6.9.1 Protocol Model

The operation of the protocol is modelled by the interaction of two (or more) protocol entities (PEs), each of which forms part of a separate application entity. Each PE implements a finite state machine. The two PEs communicate by means of the services available at their lower boundary, in such a way as to provide the services required at their upper boundary. These concepts are illustrated in Figure 6.11.

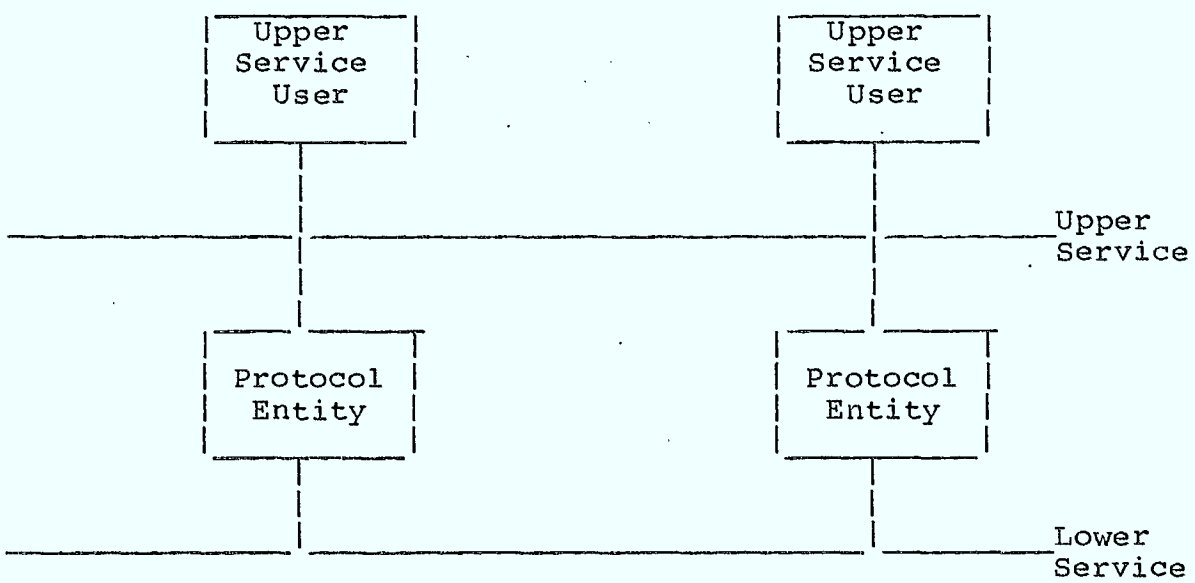


Figure 6.11 Protocol Model

The behaviour of each protocol entity is defined in terms of:

- the stimuli it receives:

  1. receipt of request or response service primitives from the upper service user;

  2. receipt of indication or confirm service primitives from the lower service provider;

  3. local events such as error indications.

- the actions it takes:

  1. issuance of indication or confirm service primitives to the upper service user;

  2. issuance of request or response service primitives to the lower service provider.

- the information it retains:

  1. information associated with the lower association endpoint; this information is lost if the lower service association ceases to exist;

  2. information associated with the upper association endpoint; this includes information about the functionality requested by the upper service user, e.g. ability to recover from errors.

6.9.2 Elements of a Protocol Specification

The following information must be provided as part of a protocol specification:

- services assumed from the lower service provider;

- identification of functions provided, including any classifications thereof;

- handling of protocol data units

- description of state information

- description of protocol behaviour
- specification of abstract syntax
- specification of transfer syntax(es)
- conformance statement
- protocol state tables
- identification of application context(s)

Optionally, a formal description using one of the techniques described in Clause 8 may be provided.

6.9.2.1 Services Assumed From Lower Service Provider

As a minimum, the lower service provider is capable of providing the services defined by the CASE, Presentation and Session kernel functional units.

Each functional unit of a protocol may have differing requirements for additional services, either in terms of other functional units of CASE and the Presentation and Session layers or in terms of other SASE. These requirements must be stated for each functional unit.

The relationship between a SASE and the CASE association and context control services is less precise than for other services. This is because the environment in which a SASE is to be used cannot always be foretold. A SASE designed originally to be a controlling SASE within one distributed application may eventually become a provider SASE in a different application.

Thus it is not possible to assume that the SASE will actually initiate an application association. It is sufficient that the SASE be capable of acquiring the proper context for its functioning when it is invoked. Any CASE service that performs this function is acceptable. The precise service that is to be invoked in a particular instance cannot be determined by the SASE; this is the responsibility of the application entity as a whole, including the user element. Similarly, when terminating a SASE, either gracefully or abruptly, it is necessary to relinquish the application context; the SASE assumes that such a service is available from CASE. From the SASE perspective, there exists a set of CASE context

acquiring services, a set of CASE context relinquishing services and a set of CASE context aborting services. This is all it assumes.

For other supporting services, a detailed explanation of the use of these services must be provided.

Clause 7 provides a more detailed overview of the services available from CASE and the lower layers, to assist in the selection process.

6.9.2.2 Identification of Functions and Protocols

A protocol specification may actually specify more than one protocol. For example, the Transport Layer protocol specification identifies five different classes of protocol. The FTAM protocol recognizes three protocols: the basic file protocol, the bulk data transfer protocol and the error recovery protocol.

Each protocol or protocol class is distinguished by the functions it performs. Many functions are unique to the application, but the following are performed by all protocols:

- mapping of service and parameters into Protocol Data Units (PDUs) (see below);

- ensuring the progress of the protocol, i.e. ensuring that an action initiated by the protocol does not take forever to complete. This is typically dealt with by the use of timers.

Among the possible additional functions that a protocol entity might perform are concatenation of PDUs and error recovery.

6.9.2.3 Handling of Protocol Data Units (PDUs)

PDUs are the units of information exchanged between peer protocol entities. PDUs typically are complex data types consisting of mandatory and optional data elements. When a PDU is generated by a protocol entity, the values of the data elements are set by the sender to:

i) reflect the value of service parameters on the primitive which caused the transmission of this PDU;

ii) reflect the state of the entity sending the PDU;

iii) correspond to literal items in the PDU definitions provided in the abstract syntax specification (see below).

PDUs are transmitted as user data of a service primitive request issued to the lower service provider.

The receiver of a PDU recognizes it on the basis of its data type definition. A sequence of data types received is a PDU of the stated type if the data is in the appropriate presentation context, if it contains all the mandatory items in the order given, and if it does not contain any items not present in the definition. A PDU must be contained entirely within a single primitive. Other constraints on the detection of a valid PDU may exist which are specific to the application protocol. Any such constraints must be specified.

An entity shall signal a protocol error if it receives a sequence of data items which does not form a defined PDU when a PDU is expected.

6.9.2.4 Description of State Information

Every protocol entity maintains a "major" state, which reflects the state of the entity's protocol machine. It is this state which governs the actions of the protocol when events occur.

In addition, a protocol entity may maintain other state information such as the association status of the lower service provider, names of P-contexts, checkpoint numbers, etc., which affect the behaviour of the entity in certain situations.

All state information maintained by the protocol entity is to be defined in terms of its purpose and its representation (e.g. an integer, a character string, etc.).

6.9.2.5 Description of Protocol Behaviour

For each service, the actions associated with each possible event are described. The actions are described in terms of the service primitives that are issued, protocol data units that are prepared and state information that is modified.

This description is provided for each protocol entity (initiating and responding entities) and for each protocol or protocol class.

6.9.2.6 Specification of Abstract Syntax

This is a specification of the composition of the PDUs in terms of an abstract syntax notation. The ISO ASN.1 syntax defined in ISO 8824 is preferred. This syntax defines complex data types in terms of a small collection of basic data types which are application independent. This syntax is described further in Clause 8.

It is necessary as part of the syntax specification to indicate which basic data types and which structured types are used.

The abstract syntax may also be the transfer syntax, in which case, it becomes impossible to negotiate alternate transfer syntaxes to suit different transfer requirements.

6.9.2.7 Specification of Transfer Syntax(es)

If the abstract syntax is specified using ASN.1, then a transfer syntax is implicitly defined by the ASN.1 encoding rules.

If the abstract syntax is not specified using ASN.1 and the abstract syntax is not also a transfer syntax, then it is necessary to define a transfer syntax. Such a transfer syntax defines the encoding of the abstract syntax into a bit stream.

More than one transfer syntax may be defined for an application protocol to suit different requirements, such as encryption or data compression.

6.9.2.8 Conformance Clause

This is a critical part of the specification as it determines the degree to which variability of protocol implementation is permitted.

This clause defines the conditions under which an implementor can claim that its implementation conforms to the standard. There are three points of view for conformance requirements:

1. the conformance requirements in a standard can be:

   (a) mandatory requirements: these must be observed in all cases;

   (b) conditional requirements: these must be observed only when the condition set out in the standard apply;

   (c) options: these can be selected to suit the implementation, so long as any requirements on which the option depends ow which depend on the option are observed.

2. the statements of conformance requirements in a standard can be:

   (a) positive: they state what must be done;

   (b) negative (prohibitions): they state what must not be done.

3. the requirements fall into two groups:

   (a) static conformance requirements;

   (b) dynamic conformance requirements.

   Static conformance requirements specify what capabilities the implementation must include. These requirements may be at a broad level, such as the grouping of functional units and options into protocol classes, or at a detailed level, such as range of values that must be supported for specific parameters or timers.

   Static conformance requirements and options can be of two varieties:

(a) those which concern the capabilities to be included in the implementation of the particular protocol;

(b) those which concern multi-layer dependencies - placing constraints on the capabilities of the lower layers of the system in which the protocol implementation resides.

The following list of static requirements is common to many applications:

The system shall:

- support at least the kernel functional unit;

- act in the role of initiator or responder or both;

- support the encoding which results from applying the basic ASN.1 encoding rules to the abstract syntax specification, for the purpose of exchanging protocol control information.

Other requirements that are application-specific would typically be included.

Dynamic conformance requirements are those requirements and options which determine how a protocol implementation may behave in instances of communication.

The following list of requirements is common to many applications:

The system shall:

- follow all the procedures relevant to each functional unit that the system claims to implement;

- support the mapping onto the lower service provider as defined in the specification.

For conformance testing, a statement of the capabilities and options which have been implemented, and any features which have been omitted, is needed so that the implementation can be tested for conformance against relevant

requirements, and against those requirements only. This statement is called the Protocol Implementation Conformance Statement (PICS).

The PICS is not part of the protocol specification; it is a document prepared by an implementor which acts as the basis for conformance testing. The conformance clause of the protocol specification states only what information is to be provided in the PICS.

In summary, the conformance clause of the protocol specification defines static conformance requirements, dynamic conformance requirements and the requirements for the PICS.

6.9.2.9 Protocol State Tables

A comprehensive set of state tables must be constructed to describe the behaviour of the protocol machine of each protocol entity as completely as possible.

The tables describe the protocol machine operation in terms of incoming events (e.g. receipt of a service primitive or a PDU) and actions. An action may consist of one or more of outgoing events (e.g. issuance of a service primitive or the sending of a PDU) and local actions (e.g. update a checkpoint counter).

Actions may be conditional on specified predicates, i.e. the action will take place only if the predicate is true. An example of a predicate is the result parameter of a confirm primitive indicating success. Otherwise, actions are unconditional, in which case they will always take place when the trigger event occurs.

A typical state table entry will specify the current state, an incoming event, a predicate, an action and a resulting state.

It may not be possible to explicitly specify all entity actions using state tables. In this case, all actions which are not explicitly specified must be described. An example of such an action is behaviour on detection of an invalid incoming event. As such a condition is possible for each state, it would be very cumbersome to include this in the tables. It is easier in this case to simply state what action is taken generally when this condition is detected.

6.9.2.10 Naming of Application Context(s) and Syntaxes

A typical application will require a single application context which satisfies all the requirements of the application. In other cases, the application is sufficiently complex that more than one application context is appropriate. An example of the latter case is the JTM application, where a different context is identified for each of the basic and full classes of protocol.

An application context is specified in terms of the service elements of the SASE which are applicable and also all CASE service elements which are required. For example, the JTM application requires the CASE CCR service elements; this requirement is implied by the JTM application context. The service elements identified in Step 2 are used for identifying the application context(s).

Each application context is assigned a unique name which is registered with the ISO registration authority for application context names. An example of such a name is ISO8832-JTM-BASIC. A more complete description of the concept of Registration Authority is provided in Clause 8.

The abstract syntax defined as part of this step must also be assigned a name so that it can be used for the identification of a presentation context. Note that presentation context names are temporary, having meaning only for the duration of a presentation connection, but abstract syntaxes have permanent names which are registered with an ISO registration authority.

Any transfer syntaxes defined for the abstract syntax must also be allocated unambiguous names.

****
*

6.9.3 Step 4 for Interactive Authorization

The protocol entity model described previously is applicable to this application. Each of the elements that the protocol specification should contain is addressed below.

6.9.3.1 Services Assumed from Lower Service Provider

As discussed in Step 2, this application uses connection-oriented supporting services. There is no requirement for any supporting Application Layer services other than CASE context acquiring, relinquishing and aborting services.

As there is no requirement in this application for interchange of substantial amounts of data, the Session services related to synchronization and activities are not needed.

In order to minimize the delay associated with authorizations, the minimum number of protocol exchanges is desirable. Thus, a full duplex form of dialogue is preferred. This form of dialogue has the additional advantage that it readily permits multiple authorizations to be in process concurrently -- there is no need to explicitly manage the turn.

In summary, the following table lists the supporting services required by this application by functional unit.

Table 6.2 Supporting Services Required by IA Application

Kernel Functional Unit:	set of CASE context acquiring services set of CASE context relinquishing services set of CASE context aborting services P-DATA P-DEFINE-CONTEXT P-DELETE-CONTEXT
Repeat Functional Unit:	P-DATA

Completion Functional Unit: P-DATA

Completion Ack Functional Unit: P-DATA

6.9.3.2 Functions and Protocols

No additional functions are provided beyond the mandatory ones of performing PDU mapping and ensuring progress of the protocol.

There are two separate but related protocols. One is the context management protocol which is responsible for acquiring and relinquishing the authorization application context. It is needed in a connection-oriented environment. The other is the authorization management protocol which in a connection-oriented environment can be invoked only once the application context has been acquired. This protocol could also be used by itself in a connectionless or store-and-forward environment.

6.9.3.3 Protocol Data Units

The following protocol data units are associated with the context management protocol. Note that PDUs are defined only for context acquisition. No exchange of information is needed for relinquishing or aborting a context. The CASE services are used directly.

- IA-INITIALIZE request
- IA-INITIALIZE response

The following protocol data units are associated with the authorization management protocol; the corresponding ISO 8583 messages are included in parentheses. Note that the names of protocol data units correspond to the service primitive which caused that PDU to be sent.

- IA-AUTHORIZE request (Authorization Request)
- IA-AUTHORIZE response (Authorization Request Response)
- IA-AUTHORIZE-REPEAT request (Authorization Request Repeat)
- IA-AUTHORIZE-COMPLETION request
  (Authorization Completion Confirmation)
- IA-AUTHORIZE-COMPLETION-ACK request
  (Authorization Completion Response)
- IA-AUTHORIZE-COMPLETION-REPEAT request
  (Authorization Completion Confirmation Repeat)
- IA-ACQUIRER-REVERSAL request (Acquirer Reversal Request)

- IA-ACQUIRER-REVERSAL response (Acquirer Reversal Request Respon
- IA-ACQUIRER-REVERSAL-REPEAT request
  (Acquirer Reversal Request Repeat)
- IA-CARD-ISSUER-REVERSAL request (Card Issuer Reversal Request)
- IA-CARD-ISSUER-REVERSAL response
  (Card Issuer Reversal Request Response)
- IA-CARD-ISSUER-REVERSAL-REPEAT request
  (Card Issuer Reversal Request Repeat)

There are no additional constraints on the interpretation of a valid IA PDU.

6.9.3.4 State Information

Conceptually, a different authorization management protocol machine is invoked for each financial transaction for which authorization is desired. Therefore, state information must be maintained for each such protocol machine plus the single context management protocol machine. Each machine is distinguished using the "system audit trail number".

The following states are defined for the IA context management protocol machine:

CLOSED: Application context has not been acquired.

OPEN: Application context has been acquired.

I-CTX-ACQUIRE-PD: Context acquiring pending, initiator; waiting for IA-INITIALIZE response PDU.

R-CTX-ACQUIRE-PD: Context acquiring pending, responder; waiting for IA-INITIALIZE response service primitive.

I-CTX-RELINQUISH-PD: Context relinquishing pending, initiator; waiting for CASE context relinquish notification.

R-CTX-RELINQUISH-PD: Context relinquishing pending, responder; waiting for IA-TERMINATE response service primitive.

The following states are associated with the authorization management protocol:

IDLE: No authorization is in progress.

I-AUTH-PD: Authorization pending, initiator; waiting for IA-AUTHORIZE response PDU.

R-AUTH-PD: Authorization pending, responder; waiting for IA-AUTHORIZE response service primitive.

AUTHORIZED: Authorization approved.

AUTH-COMPL: Authorization completion.

AUTH-COMPL-ACK: Authorization completion acknowledged.

I-ACQ-REV-PD: Acquirer reversal pending, initiator; waiting for IA-ACQUIRER-REVERSAL response PDU.

R-ACQ-REV-PD: Acquirer reversal pending, responder; waiting for IA-ACQUIRER-REVERSAL response service primitive.

I-ISSUER-REV-PD: Card Issuer reversal pending, initiator; waiting for IA-CARD-ISSUER-REVERSAL response service primitive.

R-ISSUER-REV-PD: Card Issuer reversal pending, responder; waiting for IA-ACQUIRER-REVERSAL response PDU.

No additional state information has been identified.

6.9.3.5 Protocol Behaviour

The behaviour of the context management protocol would be described separately from the authorization management. The principal purpose of this protocol is to establish the appropriate application context for the authorization management protocol and to select the functional units that are to be used.

The authorization management protocol would be described in terms of the input events and the actions that would be taken according to the current state of the protocol machine. The protocol machine behaviour is different for the initiator and responder, so each must be described separately.

6.9.3.6 Specification of Abstract and Transfer Syntaxes

ISO 8583 specifies a structure and an encoding for bank card messages. This encoding could be considered as a transfer syntax for this protocol. If no other transfer syntaxes were contemplated, then this could also be considered as the abstract syntax. Otherwise, it would be desirable to specify an abstract syntax that could be mapped into the transfer syntax of ISO 8583 and into other transfer syntaxes as well. The ASN.1 abstract syntax notation could be used for this purpose.

6.9.3.7 Conformance

The conformance requirements specified in Clause 6.9.2.8 are applicable to this protocol.

6.9.3.8 State Tables

Four sets of state tables would be constructed: one for the context management protocol - initiator; one for the context management protocol - responder; one for the authorization management protocol - initiator; and one for the authorization management protocol - responder.

6.9.3.9 Names of Application Context and Syntaxes

A name must be specified for the Interactive Authorization application context and registered with the ISO Registration Authority.

Similarly, names must be defined and registered for each abstract and transfer syntax defined for this protocol.

* *
****

6.10 Validation

6.1Ø Validation

6.1Ø.1 Validate the Protocol Design

A validation of the protocol specification should be made during the design phase of the protocol. The risk of design errors may be minimized by using a protocol design methodology as described in the preceding sections.

The validation of a protocol should address the following three criteria:

1. Verifying general properties that each protocol should satisfy, such as progress (no actions will take forever to complete), correctness (actions performed will be in accordance with the service definition), absence of deadlocks, and completeness (provision for the reception of all possible interactions);

2. Verifying that the local rules of the service definition are satisfied at each service boundary by the protocol entities defined;

3. Verifying that the end-to-end properties defined in the service definition are satisfied by the protocol entities interacting through the underlying communication service.

The following checklist identfies criteria for judging service and protocol specifications:

1. General Criteria

a) Has implementation-dependent detail been omitted?
b) Has sufficient implementation freedom been allowed?
c) Have meaningful identifiers been used?
d) Is the terminology consistent throughout the document and consistent with other OSI documents?
e) Have the real-time constraints been adequately specified?
f) Is the description well-structured?
g) Could the description be used as the basis of testing an implementation for conformance?
h) Have mandatory features, optional features, and additional options been clearly identified?

2. Service Description Criteria

a) Have only the abstract service primitive and their parameters been described?
b) Have classes of service and service options been clearly identified?
c) Has the relationship between service-access-points, connection-end-points, and connections been covered as required?
d) Does the description avoid unnecessary mechanisms to explain its externally visible behaviour?
e) Have flow control and unusual sequences of primitives been adequately described?

3. Protocol Specification Criteria

a) Have all protocol data units and their parameters been described?
b) Is the style of description compatible with that of the supported service? That is, would it be practicable to verify the protocol specification against the requirements expressed in the service descriptions?
c) Have flow control, invalid PDUs, inopportune PDUs and error handling been properly described?
d) Have legitimate errors in the underlying service been provided for?

6.10.2 Validate Protocol Implementations

It is not sufficient that a standard protocol be developed in order that successful interworking between systems be possible. It is necessary also that each system implement the protocol in a correct manner.

The term "protocol conformance assessment" is used for all activities that can be used for verifying whether a particular protocol implementaion adheres to the corresponding protocol specifications. These activities usually involve testing. (The possibility of using program proving methods is not considered further, although it could be relevant for validating security aspects of a system). If such checking is performed by an official organization against a standard reference specification, then the activity may be called "certification". The assessment activity consists of applying tests to the implementation, which is called "implementation under test", or simply IUT. The tests are qualitative or quantitative depending on their objective, that is, either checking the logical conformity to the protocol specification, or

measuring certain performance parameters such as throughput, delay, reliability, etc.

As the conformance requirements in a protocol specification may include conditional and optional requirements, it is necessary for a protocol implementation to state precisely in what sense it conforms to the specification, so that the implementation can be tested for conformance against relevant requirements, and against those requirements only. This statement is called the Protocol Implementation Conformance Statement (PICS).

The PICS should distinguish between the following categories of information which it may contain:

(a) information related to the optional and conditional static conformance requirements of the protocol itself (i.e. the capabilities of the protocol implementation both at the broad level of grouping functional units and options into protocol classes, etc., and at the detailed level of ranges of parameter and timer values supported);

(b) information related to the optional and conditional static conformance requirements for multi-layer dependencies; for example, the File Transfer protocol is permitted to use the Presentation Service directly for association establishment, or it may use CASE services. A particular implementation may support one or both alternatives. This must be stated as part of the PICS;

(c) other information which has to be specified (e.g. to assist testing) but which is not related to conformance requirements as such.

There are at least the following contexts in which protocol assessment may be applied:

(a) For product development: An organization implementing a protocol may apply an assessment procedure for debugging purposes, or for the final testing of the software product.

(b) For acceptance testing: When purchasing an implementation of a protocol, the purchasing organization may want to apply an assessment test to verify that the purchased product conforms to the specifications.

(c) For certification: An independent organization may provide the service of making impartial assessment tests on protocol implementations which are submitted by interested parties.

7. CATALOG OF SERVICES

7.1 Introduction

It is important that the application protocol designer be aware of capabilities provided by existing protocols, both at the lower layers and within the Application Layer, so that maximum advantage can be taken of these capabilities and duplication minimized.

For each layer, the applicable capabilities are described, their usage explained and any restrictions identified.

Only a brief overview is provided in this clause. The reader is referred to the appropriate ISO documents for further details.

7.2 Session Layer

7.2.1 Capabilities of the Session Layer

The connection-oriented session service provides the means for organized and synchronized exchange of data between cooperating session-service users. It provides its users with means to:

a) establish a connection with another user, exchange data with that user in a synchronized manner, and release the connection in an orderly manner;

b) negotiate for the use of tokens to exchange data, synchronize and release the connection, and to arrange for data exchange to be half-duplex or duplex;

c) establish synchronization points within the dialogue and, in the event of errors, resume the dialogue from an agreed synchronization point;

d) interrupt a dialogue and resume it later at a prearranged point.

7.2.1.1 Use of Tokens

The concept of tokens is used to control the right of users to invoke certain services. Services under token control can be invoked only when the associated token

is in the possession of the user wishing to invoke the service. Tokens can be exchanged between users to pass control as needed. Four tokens are defined:

a) the data token;

b) the release token;

c) the synchronize-minor token;

d) the major/activity token.

An example of the use of the data token is control of dialogue in a virtual terminal environment.

7.2.1.2 Functional Units

The capabilities of the Session Layer are organized into functional units. The application designer must be aware of these functional units as they form the basis for selection of session services during connection establishment.

Table 7.1 lists the functional units and the services associated with each. It also indicates which services are directly mapped through the Presentation Layer and therefore are available to the Application Layer.

Note that in most cases, services are associated with only one functional unit. The principal exceptions are the token management services which are associated with many functional units; this is a consequence of the multiplicity of tokens that are managed by the same set of services.

A description of each functional unit follows.

Table 7.1 Services associated with each session functional unit

Functional unit	Service(s)	Directly Mapped
Kernel (non-negotiable)	Session connection	no
	Normal Data Transfer	yes
	Orderly Release	yes
	U-Abort	no
	P-Abort	no
Negotiated Release	Orderly Release	yes
	Give Tokens	yes
	Please Tokens	yes
Half-duplex	Give Tokens	yes
	Please Tokens	yes
Duplex	no additional service	
Expedited Data	Expedited Data Transfer	yes
Typed Data	Typed Data Transfer	yes
Capability Data Exchange	Capability Data Exchange	yes
Minor Synchronize	Minor Synchronization Point	yes
	Give Tokens	yes
	Please Tokens	yes
Major Synchronize	Major Synchronization Point	yes
	Give Tokens	yes
	Please Tokens	yes
Resynchronize	Resynchronize	no
Exceptions	Provider Exception Reporting	yes
	User Exception Reporting	yes
Activity Management	Activity Start	yes
	Activity Resume	yes
	Activity Interrupt	yes
	Activity Discard	yes
	Activity End	yes
	Give Tokens	yes
	Please Tokens	yes
	Give Control	yes

Kernel functional unit: supports the basic session services required to establish a session connection, transfer normal data and release the session connection. This functional unit is mandatory.

Negotiated release functional unit: supports the negotiated orderly release service, which permits the party not initiating the release to accept or reject the release attempt. Without this functional unit, release cannot be rejected. This functional unit may be useful for applications where both parties to a connection may wish to send data; should one party attempt to release a connection once it has completed sending all its data, this functional unit allows the other party to refuse the release and proceed with its data transfer. The use of negotiated release is controlled by the release token.

Half-duplex functional unit: supports the normal transfer of data under the control of the data token. Only the owner of the data token is allowed to send normal data. As the data token can be owned by only one party at a time, this functional unit constrains the flow of normal data traffic to be one-way. This functional unit cannot be used in conjunction with the duplex functional unit.

Duplex functional unit: supports full-duplex transfer of normal data without token control. Therefore, transfer of normal data is possible in both directions simultaneously. This functional unit and the half-duplex functional units are mutually exclusive.

Expedited data functional unit: supports the transfer of small quantities (1 to 14 octets) of high priority data free from the token and flow control constraints of the other data transfer services.

Typed data functional unit: supports the transfer of data regardless of the availability and assignment of the data token. This form of data transfer is still subject to normal flow control; it is particularly useful when the half-duplex functional unit is selected for the transfer of protocol data units associated with presentation and application protocols.

Capability data functional unit: supports the transfer of a limited amount of user data (1 to 512 octets) while not within an activity. This functional unit can be selected only in conjunction with the activity management functional unit. Its functionality is very similar to the typed data functional unit; its exists for historical reasons.

Minor synchronize functional unit: supports the minor synchronization point service, which allows the session service user to separate the flow of normal and typed data transmitted before a minor synchronization point from subsequent normal and typed data. Its use is controlled by the synchronize-minor token.

Major synchronize functional unit: supports the major synchronization point service, which allows the session service user to confine the flow of sequentially transmitted normal, typed and expedited data in each direction within a dialogue unit. A dialogue unit is demarcated by major synchronization points and has the property that all communication within it is completely separated from all communication before and after it. The use of major synchronization is controlled by the major/activity token.

Minor synchronization points can be used within dialogue units to provide a finer degree of structuring. Figure 7.1 illustrates how a dialogue unit is structured through the use of minor and major synchornization points. Each minor synchronization point may or may not be confirmed explicitly.

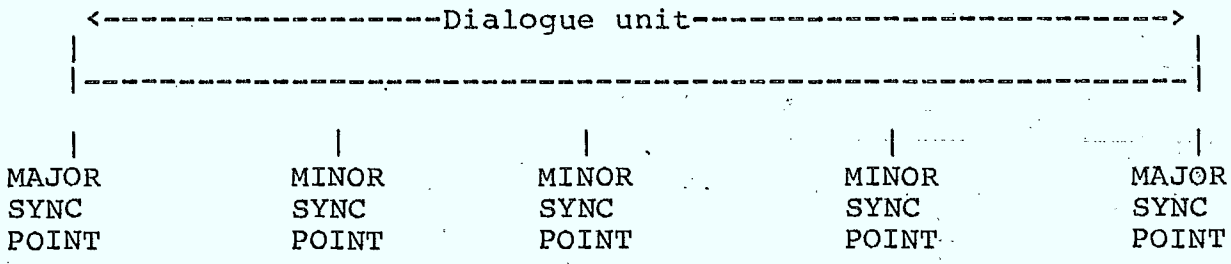


Figure 7.1 Example of a structured dialogue unit

Resynchronize functional unit: supports the resynchronize service; this service sets the session connection to a defined state, and therefore includes reassignment of tokens and purges all undelivered data. The principal uses of this functional unit are to recover from detected errors or to terminate abruptly a data transfer activity without terminating the connection.

Exceptions functional unit: supports the reporting of error conditions or unanticipated situations detected by either the session service user or session service provider.

This functional unit can only be selected in conjunction with the half-duplex functional unit; the user-initiated exception reporting service can be used only when the data token is not available to the party wishing to report an error.

Activity management functional unit: supports the services associated with activities. An activity is a logical piece of work which consists of one or more dialogue units, as shown in Figure 7.2

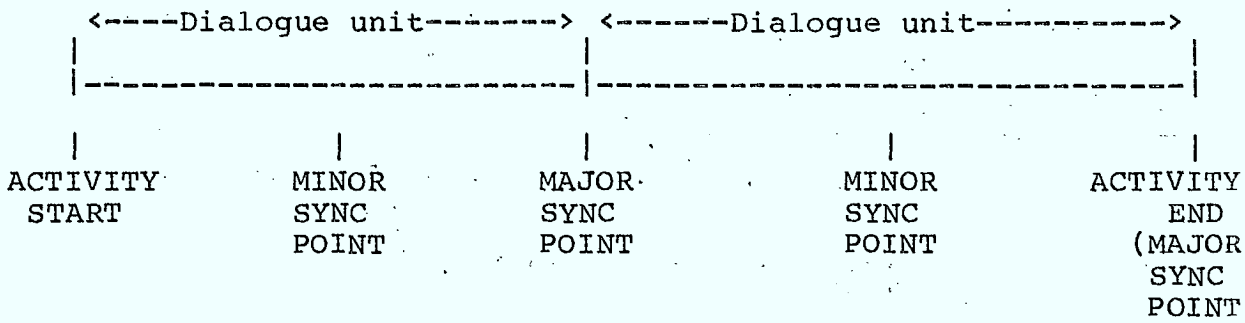


Figure 7.2 Example of a structured activity

Only one activity is allowed on a session connection at a time, but there may be several consecutive activites during a session connection. An activity may also span more than one session connection. An activity can be interrupted and then resumed on the same or on a subsequent session connection.

There are five services associated with activities:

a) the Activity Start service is used to indicate that a new activity is entered.

b) the Activity Resume service is used to indicate that a previously interrupted actity is re-entered.

c) the Activity Interrupt service allows an activity to be abnormally terminated with the implication that the work so far achieved is not to be discarded and may be resumed later.

d) the Activity Discard service allows an activity to be abnormally terminated with the implication that the work so far achieved is to be discarded, and not resumed.

e) the Activity End service is used to end an activity (and set a major synchronization point).

The use of each of the above services is controlled by the major/activity token.

7.2.2 Usage of the Session Layer

The usage of each functional unit has been outlined above in the description of each functional unit. The discussion in this section focuses on the general philosophy of usage of session services, particularly as it applies to synchronization, and on the restrictions which apply to the usage of session services.

7.2.2.1 Session Philosophy

A very important characteristic of the session service is that it deliberately does not handle resynchronization automatically. Resynchronization is initiated only by session service users and not by the session service provider. The point to which resynchronization is being done is controlled by the user.

The implication of this is that the semantics of synchronization points are determined solely by the application. The application developer must decide what the effects of major and minor synchronization points are on the application and must relate them to the service elements of the application. As an example, the file transfer protocol associates a major synchronization point with the completion of a file transfer; minor synchronization points correspond to checkpoints inserted in the transferred file.

The session service provides only the mechanisms for setting the state of a data stream to a previous state associated with a specified synchronization mark or to a completely new state.

This approach permits considerable flexibility in the use of the session services, but it places the onus on the higher layers to manage error recovery. This means that the Application Layer is responsible for determining what and when resynchronization action is to be taken and for resetting the application context (semantics) to the state associated with the specified synchronization mark.

The management of resynchronization can become more complex for the application if application associations are built containing more than one set of application semantics, e.g. JTM, CCR and FTAM. It is not always possible to know the various combinations of

application protocols that will be used, so it becomes impossible to document the relationship one standard may have with all other standards for the purposes of resynchronization.

The application designer must be aware of this problem and must apply the following rule to ensure consistency in the handling of resynchronization: If a process is within an atomic action defined by one standard, then resynchronization cannot be used to move out of that atomic action into another atomic action defined in another standard.

For example, assume an atomic action in JTM requires the moving of three files from one system to another. The JTM SASE may cause the generation of marks to control the state of the job movement. The FTAM SASE may also generate marks in each instance of the three file movements. If a problem occurred in the third file transfer (after two successful ones), then it would not be permitted to resynchronize out of the FTAM context to the JTM context. Instead, resynchronization would be constrained to the current FTAM context, permitting a clean end to the atomic action as defined by the FTAM standard. Then, having moved to the JTM context, resynchronization could be used to close the JTM atomic action, e.g. discard the successfully transferred files.

7.2.2.2 Restrictions

7.2.2.2.1 Choice of functional units

Only two restrictions apply to the session service user's choice of session functional units:

1. Either the duplex or half-duplex style of dialogue must be selected.

2. If the capability data functional unit is selected, then the activity management functional unit must be selected as well.

7.2.2.2.2 Restart type of resynchronization

When major marks are used with resynchronization, the restart type of resynchronization cannot be requested to any mark that preceded the last confirmed major mark.

7.2.2.2.3 Activities

Nested session activities are not permitted. Only one activity can be in progress at any one time.

Use of synchronization services (e.g. major and minor sync) are not permitted outside of activities.

Together, these restrictions mean that any application protocol which uses the activity management functional unit cannot be nested with any other application protocol requiring synchronization services. This is a significant limitation.

The semantics of activity identifiers are determined solely by the application and so the session service does not police their usage. This can lead to confusion if more than one outstanding activity is assigned the same identifier during a session connection. Therefore, session activity identifiers should be unique within a session connection.

7.2.2.2.4 Use of expedited data

Excessive use of this service can cause a subsequent resynchronization to be blocked. Therefore, this service should be used only very sparingly or not at all.

7.3 Presentation Layer

7.3.1 Capabilities of the Presentation Layer

The Presentation Layer provides a service that allows systems to communicate about the syntax of Application Layer information exchanges. The services allow discussion of the syntax of the exchanged information but not of the syntax within the systems. Where

differences exist between a local system syntax and the transfer syntax, mapping must occur. This mapping occurs within the Presentation Layer.

The Presentation Layer also permits access to Session Layer services. Some of these services are utilized by the Presentation Service Provider, e.g. typed data, and some are not, e.g. token management.

7.3.1.1 Functional Units

The capabilities of the Presentation Layer are organized into functional units. The application designer must be aware of these functional units as they form the basis for selection of presentation services during connection establishment.

Table 7.2 lists the functional units and the services associated with each. Only functional units not derived from session functional units are listed.

A description of each functional unit follows.

Table 7.2 Services associated with each presentation functional unit

Functional unit	Service(s)
Kernel (non-negotiable)	P-CONNECT P-U-ABORT P-P-ABORT
Context Management	P-DEFINE-CONTEXT P-DELETE-CONTEXT

Kernel functional unit: supports data transfer on whatever session functional unit data services are selected. This functional unit is always available.

Context Management functional unit: supports the definition and deletion of presentation contexts by agreement among the two service users and the service provider. A name is associated with each defined presentation context, but this name has no significance beyond the current presentation connection. This functional unit is optional and its use must be negotiated during connection establishment.

7.3.2 Usage of the Presentation Layer

7.3.2.1 Defined Context Set

All data interchanged over a presentation connection must have a presentation context associated with it. For simple applications, there might be only one such context required; such a context could be a default context known to the communicating parties by prior agreement or negotiated as part of presentation connection establishment.

When more than one context is required to be available simultaneously, then a defined context set (DCS) must be agreed to by all three parties to a communication, i.e. the presentation service provider and the two service users.

Most applications involve the transfer of both protocol-control-information (PCI) and user data. A different abstract syntax would normally required for each type of information. Hence there is a need for a defined context set when the two types of information are intermingled.

When CASE is used, then there will always be a defined context set as CASE uses its own presentation context for CASE protocol data units.

The definition of presentation contexts is normally performed during connection establishment. The addition and deletion of contexts in the course of a connection need be performed only if a particular presentation context is not required for the entire duration of the connection. An example of such a situation is a file transfer protocol which might require a different presentation context for the contents of each file being transferred. In this case, the appropriate context would be defined and deleted before and after each file transfer. The application protocol becomes responsible for these actions.

7.3.2.2 Resynchronization

When an application instigates resynchronization, the Presentation Layer is responsible for resetting the presentation context (appropriate transfer syntax). Unlike the Session Layer, the Presentation Layer tries to remember state information (the Designated Context Set) so that it can be restored if possible when resynchronization occurs.

When a resynchronization with restart is invoked, then the defined context set is restored to that in force at the specified synchronization point if the synchronization point occurred within the current connection; otherwise, then the context set is set to that that determined by the P-CONNECT service. An exception to this is when resynchronization occurs after a P-ACTIVITY-RESUME service, in which case the scope of synchronization points which may be specified is set to the beginning of the Activity, which may not have occurred within the current presentation connection.

When a resynchronization with set, an activity discard or an activity interrupt occurs, the defined context set is restored to that determined at time of connection establishment.

When a resynchronization with abandon occurs, the user may choose to restore the defined context set associated with a previous a synchronization point. Otherwise, the defined context set becomes that determed during connection establishment.

If the application does not wish to maintain the defined context set selected by the Presentation Layer, then it must explicitly delete and add presentation contexts using the available services.

7.4 Common Application Service Elements (CASE)

7.4.1 Capabilities of CASE

Common application service elements are service elements within an application entity that are of common use in the Application Layer.

There are two sets of common application service elements:

1. one set deals with the establishment of connection-oriented application associations between pairs of application entities. These service elements also deal with the management of application contexts between a pair of associated application entities. These service elements are designated as CASE Part 2.

2. a second set deals with the commitment, concurrency and recovery (CCR) aspects of defined groups of application entities that are cooperating in a distributed enterprise. These service elements are designated as CASE Part 3.

7.4.2 Functional Units

The CASE capabilities are organized into functional units. The application designer must be aware of these functional units because the definition of an application context includes the identification of CASE service elements that are applicable.

Table 7.3 lists the functional units and the services associated with each. Only CASE-specific services are identified.

Temporary Note: Table 7.3 includes the services identified in the proposed addendum to CASE Part 2. Thus, the kernel functional unit includes A-TRANSFER, which is not present in the current Kernel subset defined in ISO 8649/2. Further, the CCR services are considered as a separate functional unit, although the term functional unit is not used in ISO 8649/3.

A description of each functional unit follows.

Table 7.3 Services associated with each CASE functional unit

Functional unit	Service(s)
Kernel	A-ASSOCIATE A-RELEASE A-U-ABORT A-P-ABORT A-TRANSFER
Context Management	A-CONTEXT-DEFINE A-CONTEXT-SWITCH A-CONTEXT-DELETE
Commitment, Control and Recovery	C-BEGIN C-PREPARE C-READY C-REFUSE C-COMMIT C-ROLLBACK C-RESTART

Kernel Functional Unit: supports the establishment and release of application associations, the identification of the application context(s) which are applicable to the association, the selection of an initial application context and the transfer of user information between peer application-entities. It also provides the means for identifying the Presentation and Session

requirements for supporting the application association.

Context Management Functional Unit: supports the definition of additional application contexts, the deletion of existing defined contexts and the selection of a current application context from the set of defined contexts.

Commitment, Concurrency and Recovery Functional Unit: supports a distributed application where more than two application-entities participate and where more than one open system interconnection is involved. This functional unit provides a means for coordinating the activities on the separate interconnections; it provides services for commencing and concluding protocol exchanges and related activity on each interconnection so that the entire sequence appears to other applications as atomic (i.e. indivisible), even in the face of failures of individual applications.

This functional unit uses a two-phase commitment protocol involving a master and a number of subordinates which perform activities which are to be considered as part of an atomic action. In phase 1, the master determines whether the subordinates are prepared to carry out, i.e. to commit to, the action. It is only following offers of commitment or refusal from all subordinates that the master decides whether to proceed or not. In phase 2, all subordinates are ordered to commit or to rollback. Commitment implies completion of the individual activites while rollback implies cancellation.

An atomic action may involve a tree of activities, such that the initiation of one activity as part of the atomic action spawns another activity which in turn becomes a part of the atomic action.

Thus, a CCR user either

a) acts as a superior in relation to one or more subordinates; or

b) acts as a subordinate in relation to a single superior; or

c) acts simultaneously as a subordinate to one superior and as a superior to one or more subordinates.

The master of an atomic action is the initiator of the action and acts only in the role of superior.

The following are the main features of an atomic action:

a) commitment implies that the results produced by the atomic action have become permanent; no recovery capability exists at this level of operation for undoing the results produced once commitment has occurred;

b) a superior may order rollback to the initial state at any time prior to ordering commitment;

c) a superior may not order a subordinate to commit unless it has received an offer of commitment from it;

d) if a subordinate has offered commitment, it may not refuse an order to commit;

e) a subordinate may refuse commitment at any time up to making an offer to commit;

f) the superior orders rollback for any subordinate which has refused commitment.

7.4.3 Usage

7.4.3.1 Application Context Management

An application association has the notion of the defined application context list and the current application context.

Each application association has one defined application context list. It contains the names of all application contexts that have been agreed upon by both communicating application-entities. CASE specifies three services to manage the defined application context list: A-ASSOCIATE, A-CONTEXT-DEFINE and A-CONTEXT-DELETE.

Each direction of protocol data unit flow of an application association has exactly one current application context. The current application context must be a member of the defined application context list. The A-CONTEXT-SELECT service is used to specify the current application context for the direction of flow from the initiator of the service to its recipient peer application-entity.

In an application environment where a single application context is used, then the context management functional unit is not required. Note that CASE service elements are considered to be part of the application context of the user application and do not form a separate application context. Context acquisition is achieved via the A-ASSOCIATE service, context relinquishing is achieved via the A-RELEASE service and context aborting is achieved via either the A-U-ABORT or A-P-ABORT services.

In an application environment where more than one SASE is present then context management is needed. In this case, context acquisition, relinquishing and aborting may be achieved by different service elements, depending on circumstances. The set of context acquiring services includes A-ASSOCIATE, A-CONTEXT-DEFINE and A-CONTEXT-SWITCH. The set of context relinquishing services includes A-RELEASE, A-CONTEXT-DELETE and A-CONTEXT-SWITCH. The set of context aborting services includes A-U-ABORT and A-P-ABORT.

7.4.3.2 Commitment, Concurrency and Recovery

7.4.3.2.1 An example

An example of the use of this functional unit is a job transfer which requires the transfer of three files to different participants. All of the transfers must be successful if the job is to proceed. The job transfer application in this instance is the master and each invocation of the file transfer application is a subordinate.

Before each file transfer is initiated, the CCR C-BEGIN service is used to indicate to each file transfer subordinate that each transfer is part of a single atomic action. The file transfer subordinates then take responsibility for ensuring that the information

relating to each particular transfer is kept secure, i.e. safe from communication and application failures. They are also responsible for ensuring that resources that are affected by the atomic action (e.g. the contents of a file to be updated) are protected from concurrent access for the duration of the atomic action.

As each file is transferred, the recipient does not actually update the target file; it saves the received information in such a manner that it can survive an application failure. When the transfer is complete, the master indicates via the C-PREPARE service that it wants an indication from each subordinate of the success of the transfer. Each subordinate responds with either a C-READY if the answer is affirmative or a C-REFUSE otherwise. Depending on the answers received from all the subordinates, the master can choose to commit or rollback the transfer. If the master chooses to commit, each subordinate must then complete the writing of the transferred information to the target file. If the master chooses to rollback, the transferred information is discarded.

It is also possible for either the master or one of the subordinates to request a restart of an activity if something wrong is detected before commitment proceedings are initiated.

7.4.3.2.2 Restrictions

CCR is only meaningful within the context of a single distributed application involving coordination of multiple activities.

CCR cannot be used by an application which uses synchronization marks and resynchronization in a random manner unrelated to restart semantics.

CCR cannot be used outside of a session activity when use of the session activity functional unit has been negotiated.

Where a restart capability is visible in the service primitives of an application which uses CCR, the following restrictions apply to the combined sequence of service primitives:

a) application service primitives cannot be used within an atomic action to restart to a point in time earlier than the start of the atomic action; and

b) application service primitives cannot be used following commitment or rollback to return to a point in time earlier than the point of commitment or rollback.

Where an application service request primitive initiates an action whose completion (or failure) is signalled by an application service confirm or indication primitive, a C-BEGIN cannot be issued between the two primitives. This restriction prevents ambiguity over the precise point at which the atomic action starts.

7.5 Specific Application Service Elements

As part of the process of developing an application service and protocol, it is necessary to evaluate the suitability of existing services and protocols.

The following discussion describes the general characteristics of existing and proposed application standards.

7.5.1 File Transfer, Access and Management (FTAM)

The aim in the standardization of a file service is to allow the open interconnection of file users who wish to transfer, access, or manage files with elements which behave as if they store files. Anything which appears as a system for interconnection purposes, and conforms to the specified file protocols in the role of filestore provider, is considered to provide a filestore.

7.5.1.1 Functional Units and Service Classes

The FTAM standard has many capabilities organized into functional units, as shown in Table 7.4

Table 7.4 Services associated with each FTAM functional unit

Functional unit	Service(s)
Kernel	Association establishment Association termination (orderly) Association termination (abrupt) File selection File deselection File open File Close
Read	Read bulk data Data Unit transfer End of Data transfer End of transfer Cancel data transfer
Write	Write bulk data Data Unit transfer End of Data transfer End of transfer Cancel data transfer
File Access	Locate file access data unit Erase file access data unit
Limited File Management	File creation File deletion Read attributes
Enhanced File Management	Change attributes
Grouping	Beginning of grouping End of grouping
Recovery	Regime recovery Checkpointing Cancel data transfer
Restart data transfer	Restarting data transfer Checkpointing Cancel data transfer

Three service classes are identified which group functional units according to the type of file processing associated with each class:

File transfer class is intended for the transfer of entire files

File access class is intended for the manipulation of parts of files, e.g. individual records.

File management class is intended for operations of files as a whole.

7.5.1.2 Usage

The FTAM protocol can be used as a vehicle for reliable transfer of large amounts of information between systems. It is useable for smaller amounts as well, but the considerable amount of dialogue involved in preparing for a transfer (connection establishment, file selection, file open, read or write request) makes FTAM less attractive in this case.

For applications which do not require all the dialogue associated with the file semantics of FTAM, it may be worthwhile considering adapting the bulk transfer protocol aspects of FTAM which are specified as a separable protocol.

The virtual filestore model used by FTAM provides for the representation of complex file structures. Thus the FTAM concepts and protocol are applicable to many applications requiring database access.

7.5.2 Job Transfer and Manipulation (JTM)

The purpose of the job transfer and manipulation standard is to provide a set of communication-related services which can be used to perform work in a network of interconnected open systems. This work can include the running of traditional jobs.

The JTM protocol covers not only the movement of job-related data (input and output) between open systems, but also provides for the movement of data concerned with monitoring job-related activity, and for controlling and manipulating the progress of this activity.

The JTM standard uses the concept of a work statement as the data structure describing a distributed job.

Two classes of service are identified: basic and full.

Table 7.5 lists the services available in each class. The full class supports all services.

Table 7.5 Services associated with each JTM service class

Service Class	Service(s)
Basic Class	J-INITIATE-WORK J-INITIATE-WORK-MAN J-DISPOSE J-GIVE J-END-SIGNAL J-STATUS J-KILL J-STOP J-MESSAGE
Full Class	J-INITIATE-TCR-MAN J-INITIATE-REPORT-MAN J-ENQUIRE J-HOLD J-RELEASE J-SPAWN

7.5.2.1 Usage

The JTM standard supports a set of related interactions between a controlling entity (called the initiation agency) and many other entities (variously called source, sink and execution agencies). To do this, it needs to coordinate multiple associations. Hence this application is more complex than the FTAM one.

As a consequence of the distributed nature of this application, JTM can and does make use of the CCR capabilities described earlier.

Thus JTM provides the framework for a wide range of applications requiring centralized control of many activities.

7.5.3 Virtual Terminal Protocol (VT)

The virtual terminal basic class standard supports the interactive transfer and manipulation of graphic data by virtual terminal users. This graphic data is structured in a manner which models the class of character-box oriented terminals. The structuring of graphic elements is limited to images consisting of character-box graphic elements arranged in a one, two or three dimensional array.

The transfer and manipulation of graphic data takes place within a virtual terminal environment defined by a set of parameters. Profiles are used to represent sets of predefined parameters as an aid in negotiating the characteristics of a virtual terminal environment. They can be used to represent the characteristics of commonly encountered virtual terminals.

The VT service permits the transfer and manipulation of data in a way that is independent of the internal representation of information used by each virtual terminal user; provision is made to control the integrity of the communication between users through the assignment of access rights to different sets of objects (e.g. a virtual screen and a virtual keyboard).

Both synchronous and asynchronous modes of operation are supported.

7.5.3.1 Service Subsets

Three subsets are identified. They differ only in the level of negotiation available in each subset. Each subset is a superset of the next lower subset.

Table 7.6 lists the service subsets and the service facilities available for each.

Table 7.6 Facilities associated with each VT service subset

Service subset	Facilities
Kernel	Association establishment Association termination Data transfer Delivery control Token management
Profile switch negotiation	Switch profile negotiation
Multiple interaction negotiation	Multiple interaction negotiation

7.5.3.2 Usage

This application is essentially a stand-alone one which is not amenable to incorporation within other applications.

The VT negotiation mechanism is sufficiently powerful that the standard is not restricted to the virtualization of commonly encountered terminal characteristics. If an application required specialized characteristics, yet remained faithful to the architectural framework of this standard (i.e. character box graphic information in one, two or three dimensional arrays), then it could be accommodated by this standard.

7.5.4 Message Transfer (MT)

The Message Transfer service supports reliable store-and-forward transfer of messages. It delivers messages to one or more recipients within a defined time period and, where required, performs syntax transformation on the contents of messages.

This service was designed for use as part of the ISO Message Oriented Text Interchange System (MOTIS) which is an extension of the CCITT MHS. MOTIS identifies two sublayers, the interpersonal messaging sublayer and the message transfer sublayer. The former is specific to the interchange of electronic mail messages between

users while the latter is a general purpose sublayer. It is the service and protocol associated with the message transfer sublayer that is of potential interest to application developers.

7.5.4.1 Capabilities of the MT Service

The MT service provides the following capabilities:

1. the submission of a message for transfer to one or more recipients. The originator of the message can control various aspects of message processing such as message priority, the need for notification of non-delivery, deferred delivery, etc.

2. the determination of whether or not a message could be delivered to one or more recipients if it were submitted (probe).

3. the delivery of a message to a specified recipient, along with indications of message characteristics.

4. non-delivery notification to the originator in the event that a message is not delivered to an intended recipient.

5. delivery notification in the event that the originator requested explicit confirmation of delivery.

6. the ability to request cancellation of a message previously submitted for deferred delivery.

Table 7.7 lists the services that are available.

Table 7.7 Services associated with Message Transfer

Service
Submit
Probe
Deliver
Notify

7.5.4.2 Usage

The Message Transfer protocol uses the session activity services to achieve reliable transfer of messages. Hence, any user of this service must be aware that additional synchronization services such as CCR cannot be used in conjunction with this protocol.

7.6 Other OSI Aspects

In addition to existing services and protocols, there are a number of areas where ongoing OSI work is relevant to the application protocol developer.

7.6.1 Connectionless Data Transfer

Connectionless data transfer is the transmission of a single unit of data from a source to one or more destinations without establishing a connection.

7.6.1.1 Service Characteristics

In contrast to connection-oriented transfer, an instance of the use of connectionless service does not have a clearly distinguishable lifetime. In addition it has the following fundamental characteristics:

a) it requires only a pre-arranged association between the peer-entities involved which determines the characteristics of the data to be transmitted, and no dynamic agreement is involved in an instance of the use of the service;

b) all the information required to deliver a unit of data - destination address, quality of service, options, etc. - is presented to the layer providing the connectionless service together with the data, in a single service access which is unrelated to any other access.

Each layer supports a single service, namely UNIT-DATA, which permits the transfer of a unit of data between peer-entities.

In general, a connectionless service does not guarantee that units of data will not be lost, corrupted or delivered out of sequence. However, it is possible architecturally to incorporate segmentation, sequencing, acknowledgement and error handling functions within layers to lessen the probability that undesirable events will occur.

7.6.1.2 Usage

At present, services and protocols for connectionless data transfer exist only for the Transport layers and below. Further, existing protocols do not include any error handling functions. The consequence of this is that it is not possible to develop connectionless application layer protocols at this time in the absence of adequate supporting services.

Were adequate supporting services available, then the connectionless form of data transfer would be appropriate for applications which transferred small amounts of isolated data. A transaction-oriented request-response application would be one candidate.

7.6.2 OSI Security

Clause 5.5.8 of this document has identified the principal features of the OSI Security Architecture. Table 5.1 has listed the possible layers where various security services could be provided.

7.6.2.1 Application Layer Security Services

The only security services that are permitted in the Application Layer are access control and limited traffic flow security.

Limited traffic flow security is provided by use of traffic padding, which must be used in conjunction with a confidentiality service at a lower layer.

The access control service is provided by appropriate specific access control mechanisms.

Two other services involve the Application Layer. One is a non-repudiation service based on a notarization mechanism, whereby a trusted relay at Layer 7 acts as a third-party notary which resolves disputes between communication partners. This service operates in conjunction with the Presentation Layer which handles syntactic representation aspects of the service.

The other service which involves the Application Layer is an application-to-application acceptance of data service, where the Application Layer provides a mechanism to indicate proper receipt of data, e.g. after safe storage of the received data, which is used in conjunction with a Presentation Layer signature service.

7.6.2.2 Usage

The developer of an application which has security requirements must take the OSI security architecture into account so that the appropriate security functions are properly situated.

The developer must also be aware that no existing OSI protocols have incorporated security features yet. This implies that any near-term development of secure applications may require extensions to lower layer protocols which will likely render them non-standard.

7.6.3 Multi-peer Data Transmission (MPDT)

Multi-peer transmission can be described as any member of a group of entities being able to communicate with one or more members of the group.

Applications which require this capability include electronic mail, distributed database processing, key management, etc.

The aim of multi-peer services and protocols is to enable savings to be achieved in bandwith and/or elapsed time by permitting the data to be transmitted only once from a source.

7.6.3.1 Types of MPDT Services

Two types of multi-peer data transmission services have been identified:

Type 1: The MPDT is an application service which provides the distribution of data on demand of an application entity, based on existing connection-oriented or enhanced connectionless services. In this type, the multi-peer capability is introduced only in the Application Layer or in connectionless services. An example of this type of multi-peer capability is the Message Handling Systems application where a multi-peer capability, i.e the ability to send a message to many recipients, is handled via Application Layer mechanisms which use existing point-to-point lower layer services.

Type 2: The MPDT is a multi-peer application service based on multi-peer or point-to-point, connection-oriented or connectionless services. In this type, the multi-peer capability is introduced in any layer if useful. If the lower layers provide a MPDT service, the role of the layers above the Transport Layer is to initiate and control the multi-peer data transmission service, while the role of Transport and lower layers is to provide the service efficiently.

The mechanisms required for multi-peer data transmission in the upper layers are different in nature from those required in and below the Transport Layer. In the lower layers, they may depend on the characteristics of the technologies being used.

In the upper layers, the requirement is defined in detail by the application but in general the need is to permit the transmission of protocol data units carrying identical data to several destinations. This may be achieved most efficiently if identical protocol control information is used for each destination, and in particular if it is possible to establish identical transfer syntaxes to each destination and to ensure that protocol data units at every layer are made identical.

7.6.3.2 Implications for Protocol Design

When an application incorporates MPDT, a number of issues must be addressed:

Naming and Addressing: The invocation of an MPDT service requires the use of one of the following:

- a group name
- a group address
- a list of Presentation service-access-point addresses

If group names or addresses are used then mechanisms are required for managing them. This could be a directory management function..

Connection Establishment: In current point-to-point protocols, the result of negotiation during connection establishment is essentially chosen by the responder. For MPDT, it may be necessary for the originator to choose the result of the negotiation. This implies a three-way handshake.

Connection Release: The issue here is how to release the multi-endpoint connection. The connection may need to be released as soon as one end-system choses to break it or it may be permissible to continue the connection between the end-systems after one or more choose to disconnect.

Transfer of Data: Flow control becomes an issue when one or more end-systems cannot receive data at the same rate as the others. Two possible approaches are:

a) drop the slow end-system from the multi-endpoint connection, temporarily or permanently;

b) adjust the transmission rate so that it is not too fast for the slowest end-system.

Case a) corresponds to a type of application where end-to-end delays are most important, e.g. distribution of an important message.

Case b) corresponds to a type of application where bandwidth saving or synchronization of end-systems is most important, e.g. disribution of database updates.

Error Handling: The issue here is how to deal with the situation where one or more end-systems does not correctly receive a protocol data unit.

Again there is a choice of actions:

a) ignore the error and continue transmissions;

b) retransmit the incorrectly received protocol data unit, either to all connection end-points or just to those which did not correctly receive the protocol data unit.

c) if the protocol data unit is to be retransmitted, then the end-systems requiring retransmission are effectively slowing their reception rate, leading to the flow control problem discussed above.

Quality of Service: The maintenance of quality of service in a MPDT environment becomes a more complex issue. Among the considerations that must be taken into account are active group integrity, delivery reliability, sequence preservation, and the handling of confirmed services.

Management: One of the additional management functions is to provide procedures for joining, leaving and managing the membership of a group involved in MPDT communication.

7.6.4 Management Information Services (MIS)

Management Information Services are conceived of as a set of application services and associated System Management Application Processes (SMAPs) used to exchange management information and management control sequences.

The following functional areas are supported by MIS:

1. Directory Management

2. Fault Management

3. Accounting Management

4. Security Management

5. Configuration Management

6. Performance Management

7.6.4.1 Directory Management

Directory management is concerned with the provision and maintenance of directory information.

The following functions are available:

- Create Object Name
- Delete Object Name
- Add Alias
- Delete Alias
- List Aliases of
- List Property Names
- List Property Values
- Add Property
- Delete Property
- Change Property Value
- List Group Members
- Add Group Member
- Delete Group Member
- Is Member
- Find Name

7.6.4.1.1 Implications for SASE Development

The directory service is not generally accessed directly by SASEs. In the normal course of events, a SASE entity is provided with an application entity title of a peer SASE entity; this information is passed on to CASE which initiates the directory lookup function.

7.6.4.2 Fault Management

Fault management is the management of the abnormal operation of the system.

Faults manifest themselves as errors in the operaion of the system, that is, errors are the detection mechanism for faults.

The three principal aspects of fault management are fault detection, fault diagnosis and fault correction.

Fault detection may be achieved by reporting error conditions, by performing confidence tests or by checking for exceeded threshold values.

Fault diagnosis can be done by analyzing symptoms (i.e. past events) or executing diagnostic tests.

Fault correction may involve the interchange of repair action commands and reports.

7.6.4.2.1 Implications for SASE Development

There is no need for a designer of a SASE protocol to incorporate explicit fault management functions within the SASE unless the requirements of the SASE go beyond those provided by the standardized fault management services.

There will be in general a requirement for a SASE implementation to incorporate some fault detection capabilities for use in conjunction with the fault management facilities. In particular, trace facilities, consistency checks and threshold counters should be included.

7.6.4.3 Accounting Management

Accounting management provides mechanisms for communicating information between open systems relating to the monitoring and control of charges for use of communications resources and to the provision of tariff information.

Accounting management recognizes two levels at which accounting information is made available:

a) at the Network Layer which accounts for the use of the communications medium;

b) at the Application Layer which accounts for the above charges together with those incurred in the use of the end-system resources in effecting the communication.

The monitoring of charges incurred includes the capabilities for billing at the end of an accounting period, for providing advice upon termination of a specific instance of communication and for seeking or providing advice about a previous instance of communication.

The controlling of charges incurred includes the capabilities for a user to identify a payment limit, for a service provider to signal an event when the limit is about to be exceeded and to signal termination of a communication when the limit is exceeded.

The provision of tariff information includes a capability for a user to enquire about costs before initiating a communication.

7.6.4.3.1 Implication for SASE Development

A SASE should not include any accounting information within its protocol exchanges unless the standardized capabilities are inadequate.

A SASE implementation will require the capability to collect accounting information and to make it available to the local accounting subsystem.

7.6.4.4 Security Management

Security management is concerned with the management aspects of the provision of security services.

Requirements exist for authentication management, access control management, key management, handling of security audit trails and events, and access control to the Management Information Base (the repository of all management-related information).

Authentication management involves the distribution of passwords to entities involved in authentication control.

Access control management involves the distribution of passwords, access control lists and capability lists and their updating.

Key management involves the distribution of encryption keys.

The handling of security audit trails and events includes remote collection of audit records and enabling and disabling audit trail logging.

MIB access control is a special instance of access control that is specific to management.

7.6.4.4.1 Implications for SASE Development

Security management has no direct implications for SASE development.

7.6.4.5 Configuration Management

Configuration management is concerned with the determination and control of the logical and physical configuration of a system.

The following functions are included as part of configuration management:

a) automated collection and reporting of information about the system state, e.g. what functionality is available and whether it is active or inactive;

b) control of the physical configuration; this includes activation and deactivation of physical devices;

c) control of the logical configuration; this includes principally the activation and deactivation of software resources;

d) software distribution control, to ensure that all users are using consistent versions of software.

7.6.4.5.1 Implications for SASE Development

There are no direct implications for SASE development.

7.6.4.6 Performance Management

Performance management is the control and evaluation of statistical information derived from within an open system.

The aspects of this function are:

a) collect the system statistics;

b) control the collection of system statistics;

c) store the system statistics and their histories;

d) analyse the system statistics;

e) present the system statistics.

7.6.4.6.1 Implications for SASE Development

This function has no direct implications for protocol development.

As in the case of fault management, there is a need for an implementation to incorporate counters, etc. to provide the statistics needed by this function.

8 PROTOCOL DEVELOPMENT TOOLS

This clause describes various tools and mechanisms that the application protocol developer should be aware of.

8.1 Registration Procedures

The successful application and use of OSI standards requires that certain objects and/or that unique names be registered with some authority.

An example of this requirement is the registration of abstract and transfer syntaxes. OSI Application and Presentation Layer entities rely on a protocol-based negotiation mechanism to reach agreement with their peer entities on the subject of their communication and its representation during their dialogue. This negotiation is based on the ability to unambiguously identify a syntax known to both peer entities, and is achieved through reference to a reserved or registered name for that syntax.

To achieve successful communication between open systems, standards are required to specify: the form or structure of these names; the extent of domains in which they are unique; the authority or procedures required to register the names of newly defined syntaxes.

This requirement applies to the following types of names:

- transfer syntaxes
- abstract syntaxes
- application titles
- service-access-point addresses
- application contexts
- document types

Specific Registration authorities responsible for individual naming domains will be registered with and coordinated by the general OSI Registration Authority.

Thus the process of registering a name for an object requires first that a specific Registration Authority be established, then that a proposal be submitted to that Registration Authority to register the proposed name.

In the case of abstract and transfer syntaxes, a single Registration Authority could be established to handle both.

As an illustration of the type of information that is associated with a register entry, a register of abstract syntaxes would contain the following information for each registered syntax:

a) an unambiguous name for the abstract syntax (assigned by the Registration Authority) and a list of any registered aliases;

b) a reference to the definition of the abstract syntax;

c) references to those registered transfer syntaxes that are capable of supporting the information representation requirements of the abstract syntax;

d) the identity of the organization that requested the registration of the object;

e) the date of registration;

f) status information (e.g. version and obsolescence data);

g) variable features of the registered object (i.e. options of one or more forms).

A register entry for transfer syntaxes would contain similar information, with the references to transfer syntaxes in c) above replaced with references to registered abstract syntaxes that the transfer syntax supports.

8.2 Formal Description Techniques (FDTs)

Different methods may be used for writing service and protocol specifications; they range from the use of natural language to rather formal mechanisms. The use of natural language has the drawback that it gives the

illusion of being easily understood, but leads to lengthy and informal specifications which often contain ambiguities and are difficult to check for completeness and correctness.

A formal protocol specification, on the other hand, is less prone to the difficulties of a natural language specification.

A formal description technique (FDT) is the vehicle for such formal specifications.

The main objectives to be satisfied by an an FDT are that it should be:

a) expressive: an FDT should be able to define both the protocol specifications and the service definitions of the seven layers of OSI;

b) well-defined: an FDT should have a formal mathematical model that is suitable for the verification of these specifications and definitions. The same model should support the conformance testing of implementations.

c) well-structured: an FDT should offer means for structuring the description of a specification or definition in a manner that is meaningful and intuitively pleasing. A good structure increases the readibiity, understandability, flexibility, analysability and maintainability of system descriptions.

d) abstract: there are two aspects of abstraction that an FDT should offer:

   1. an FDT should be completely independent of methods of implementation, so that the technique itself does not provide any undue constraints on implementors.

   2. an FDT should offer the means of abstraction from irrelevant details with respect to the context at any point in a description.

   Abstraction can reduce the local complexity of system descriptions considerably. In the presence of a good structure, abstraction can help even further to reduce the complexity of descriptions.

ISO is currently developing two such FDTs, ESTELLE and LOTOS.

8.2.1 ESTELLE

Estelle is a formal description technique based on an extended state transition model. Estelle may be viewed as a set of extensions to ISO Pascal level Ø which allows the components of a data communications protocol to be modelled as one or more modules each of which is specified as an extended finite state machine.

A system described by an Estelle specification consists of a set of cooperating modules. The purpose of each module specification is to define the behaviour of the module as observable at its interaction points, i.e. the points where the module interacts with other modules and its environment.

The behaviour of each module can be defined by a state transition model, consisting of input and output interactions, states and transitions.

Since finite state diagrams or equivalent methods often lead to very complex representations for specifications, the finite-state model is extended with the addition of variables to the states, parameters to the interactions, and priorities to the transitions. This includes the introduction of types which are used for variables and parameters as well as for modules. This approach combines the simple concept of states and transitions with the power of a programming language.

Estelle permits refinement of module structure into a set of nested modules. Thus a specification consists of a tree of modules organized into a hierarchy.

Estelle is to some extent an implementation-oriented description technique in that it does describe the internal behaviour of an idealized implementation of the protocol. While an implementation which claims to conform to an Estelle specification is required only to have externally observable behaviour consistent with the specification, in practice it may be difficult to verify such behaviour if an implementation is radically different internally from the idealized structure of the specification.

8.2.2 LOTOS

LOTOS (Language of Temporal Ordering Specification) is based on the concept that protocol systems can be described by defining the relation between events in the externally observable behaviour of a system.

The formal mathematical model of LOTOS is based on a modification of the Calculus of Communicating Systems (CCS), which was developed at the University of Edinburgh. CCS provides a powerful analytical technique for concurrent systems.

In LOTOS, distributed systems are modelled as a set of communicating processes. A system as a whole is a single process that may consist of many interacting subprocesses. These subprocesses may in turn be refined into subprocesses, etc., so that a specification of a system in LOTOS is essentially a hierarchy of process definitions.

The static picture of a process can be imagined as that of a black box that is capable of communication with its environment. The mechanisms inside this box are not observable, therefore in principle not part of the model. The way in which a process may be described is by specificatin of its ability to communicate with its environment. A process communicates with its environment by means of interactions. The atomic form of interaction is an event. An event is a unit of synchronized communication that may exist between two processes that can both perform that event.

The behaviour of a process may be thought of as having a tree-like structure. The root of the tree represents the initial state of the process. The edges in the tree are labelled by the names of events. In this way, the labels on the outgoing edges of each node represent possible next steps of the process. The tree structure thus represents process behaviour as a sequence of possible choices ordered in time according to the depth of the nodes in the tree.

The approach taken in LOTOS, viz. the description of behaviour in terms of composition principles that reflect intuitive notions about the way in which systems are structured (sequence, choice, parallellism, disruption) enables meaningful modularity in the definition of very complex systems.

The CCS model on which LOTOS is based focuses on the dynamic aspects of process behaviour and does not define the mechanism to use for the representation of data. LOTOS uses ACT ONE as the mechanism for defining data types and values; ACT ONE is based on the theory of abstract data types. It provides capabilities similar to the type definition capabilities of Estelle (i.e. Pascal).

8.2.3 Checklist for an FDT-based Specification

Clause 6 included a checklist for ensuring the quality of a protocol specification. When a formal description technique is used, the following questions must be answered as well:

1. Does the formal description form an essential part of the standard or is it provided only for guidance?

   It is very important to have a clear understanding of the status of the formal description. Ideally there should be no discrepancies between the text and the formal description, but because this is very hard to achieve in practice it is important that the reader knows which takes precedence. If the formal description is provided only for guidance, it cannot define conformance requirements.

2. Is the FDT well-defined, stable and properly referenced?

3. If the formal description defines requirements, does it include all the requirements of the standard?

   If not, it must be clearly stated that the text includes requirements which are not covered by the formal description and these additional requirements should be clearly identified.

4. If the formal description defines requirements, does it also define an allowed way of implementing some aspects of the protocol?

   If so, but there is intended to be freedom for the implementor to implement those aspects in some other way, then this constitues overdefinition. This is all too common in formal descriptions and

creates difficulties in relation to conformance. If the formal description is an essential part of the standard, then text must be provided to qualify it, indicating where such over-definition exists and what the real requirements are.

The problem usually arises because the formal description describes teh internal behaviour of an idealised implementation, rather than just the observable external behaviour that is required. It is only the observable external behaviour which can be tested, and therefore it is only this which should constitute requirements for conformance purposes. It may well be that a different FDT should be used for defining the requirements from that used to provide guidance to implementors.

8.3 Abstract Syntax Notation One (ASN.1)

In the lower layers of the OSI Reference Model, each user data parameter of a service primitive is specified as the binary value of a sequence of octets.

In the Presentation Layer, the nature of user data parameters changes. Application Layer standards require the presentation service user data to carry the value of quite complex types, possibly including strings of characters from a variety of character sets. In order to specify the value which is carried, they require a notation which does not determine the representation of the value. This is supplemented by the specification of one or more encoding rules which determine the value of the Session Layer octets carrying such Application Layer values (this is the transfer syntax). The Presentation Layer negotiates which transfer syntaxes are to be used.

ASN.1 defines a number of built-in types and associated values, and provides type constructors which allow the construction of complex data structure types (called structured types) and corresponding complex data values from these basic built-in types. Common built-in types include boolean, integer, bit string and octet string. The common constructors are sequence, set and choice. Various types of character string are supported, including ISO 646 strings, numeric character strings, printable character strings (a basic 74-character set), teletex character strings, videotex character strings and graphic character strings. The latter permits the use of any character set in the ISO Register of Character Sets, as established by ISO 2375.

An important objective of ASN.1 is to support the representation of Application and Presentation Layer protocol data units and to facilitate encoding. For this reason, it provides a means for associating identification tags with data types. This is in contrast to the data structure definition facilities of Estelle and LOTOS, which were not intended expressly for PDU representation.

8.4 Conformance Testing Methodology

The objective of OSI will not be completely achieved until systems can be tested in order to determine whether they conform to the relevant protocol standards.

Standard test suites should be developed for each protocol standard for use by a supplier or implementor in self-testing, by the user, or by the carrier or other third-party tester. This should lead to comparability and wide acceptance of test results produced by different testers, and thereby minimise repeated conformance testing of the same system.

The standardization of test suites requires international definition and acceptance of a common testing methodology and appropriate testing methods. Such a common framework and methodology is being developed by ISO.

8.4.1 Objectives of Conformance Testing

In principle, the objective of conformance testing is to establish whether an implementation being tested conforms to the specification in the relevant standard. Practical limitations make it impossible to be exhaustive, and economic considerations may restrict testing still further.

Three types of testing are identified, acording to the extent to which they provide an indication of conformance:

- basic interconnection tests which provide prima facie evidence that an implementation under test (IUT) conforms;

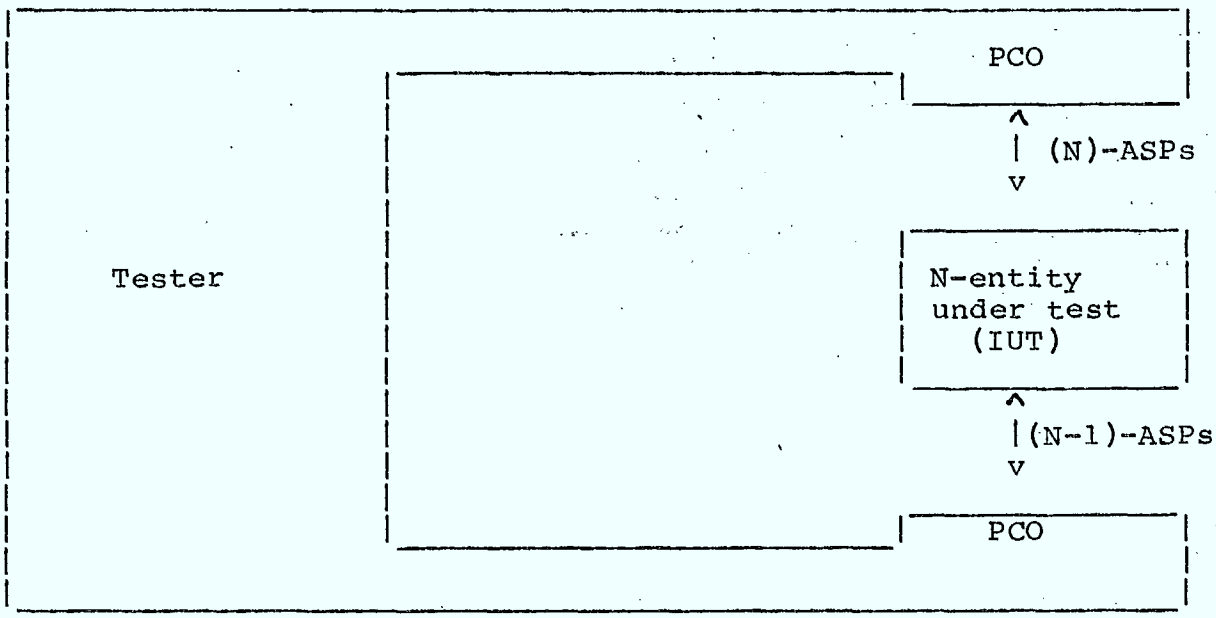
- conformance tests which endeavour to provide as comprehensive testing as possible over the full range of requirements specified by the standard;

- conformance resolution tests which provide a definite yes/no answer in the context of specific conformance issues. Such tests serve a diagnostic function, and are typically used when the conformance tests indicate suspicious behaviour of the system, but are not capable of determining precisely whether a problem exists.

8.4.2 Abstract Testing Methodology

The essential characteristic of the test methodology is that it examines the externally observable behaviour of a protocol entity. The OSI protocol standards define allowed behaviour of a protocol entity in terms of the protocol data units (PDUs) and both the abstract service primitives (ASPs) above and below that entity.

Figure 8.1 illustrates the conceptual testing architecture.



Legend: PCO : point of control and observation
ASPs: abstract service primitives
IUT : implementation under test

Figure 8.1 Conceptual Testing Architecture

Three test methods are identified. They differ in the type of mechanism required for control and observation of the upper and lower interfaces of the IUT.

1. Local test methods which use control and observation of the ASPs directly above and below the IUT;

2. Distributed test methods which use control and observation of ASPs directly above the IUT and control and observation of the (N-1)-ASPs as seen by a remote tester acting over the (N-1) service provider. Thus the tester intelligence is disributed over the system under test and a remote system.

3. Remote test methods which use control and observation of the (N-1)-ASPs as seen by a remote tester acting over the (N-1) service provider. The ASP activity above the IUT is unspecified.

8.4.3 Implications for Application Protocol Testing

With the Application Layer, there is no layer service above it which can be controlled or observed via (N)-ASPs at a service-access-point (SAP). However, for application protocols specified according to the methodology specified in this document, there is a defined service with service primitives, some or all of which may be controllable or observable. If they are, then the test methods can be applied by using these primitives rather than (N)-ASPs, but if they are not then only the remote test methods can be used.

The extent to which control and observation of application service primitives will be possible will vary from one application to another and may be different at the two sides because of inherent asymmetry in the application. For example, what can be controlled and observed for an FTAM initiator will be rather different from what can be controlled and observed for an FTAM responder (i.e. the filestore end).

Some application protocol standards might state conformance requirements which go beyond pure protocol behaviour. To test such requirements, special application protocol dependent test methods may be required to complement the test methods described in the ISO test methodology. For example, it may be necessary to inspect the contents of a filestore by local means before and after certain file transfer tests.

Static conformance requirements for syntax support should be included in application protocol standards. However, the syntax support can be considered to be implemented by the presentation entity. Thus, in order to test all the conformance requirements in the application protocol standard it would be necessary to test the application entity in combination with the presentation entity. Similarly, it would be necessary to test the presentation entity in combination with the application entity in order to test the presentation entity in the context of its use with that application.

8.5 Analysis Tools

Many methods and tools have been developed for similar fields of application and may be useful for protocol development.

The following tools may be helpful during the specification design stage, during the specification validation stage and finally during protocol implementation testing.

- simulators for existing specification languages
- simulation languages
- tools for logic reasoning (program providng and Abstract Data Types)
- tools using Artificial Intelligence (e.g. terminology, association formal-informal)
- tools exploiting Data Base Management Systems (cross-referencing, etc.)
- tools for finite state machine and Petri Nets analysis
- tools for syntax and static semantic consistency checking
- tools for checking observed interaction sequences
- tools for data flow analysis
- tools for determining the coverage of tests

ANNEX A - LIST OF REFERENCES BY TOPIC

NOTE 1 - References preceded with an asterisk (*) are presently at the stage of draft; publication expected in due course.

Upper Layer Architecture

ISO TC 97/SC 21 N 876 Architectural Detail of the Upper Three Layers of OSI

ISO TC 97/SC 21 N 539 Proposed Application Layer Structure

ISO TC 97/SC 21/WG 1 N 79 A more precise definition of basic OSI concepts

OSI Reference Model

ISO 7498/1 Information processing systems - Open Systems Interconnection - Basic Reference Model.

*ISO 7498/2 Information processing systems - Open Systems Interconnection - Security Architecture.

*ISO 7498/3 Information processing systems - Open Systems Interconnection - Naming and Addressing.

*ISO 7498/4 Information processing systems - Open Systems Interconnection - OSI Management Framework.

*ISO 7498 DAD1 Information processing systems - Open Systems Interconnection - Addendum to Reference Model for Connectionless Operation.

*ISO 7498 DAD2 Information processing systems - Open Systems Interconnection - Addendum to Reference Model on Multi-peer Data Transmission.

Service Conventions

*ISO TR 8509 Information Processing Systems - Open Systems Interconnection - Service Conventions.

Application Layer Standards

*ISO DIS 8571 Information Processing Systems - Open Systems Interconnection - File transfer, access and management. Parts 1 to 4.

*ISO DIS 8831 Information Processing Systems - Open Systems Interconnection - Job Transfer and Manipulation concepts and services.

*ISO DIS 8832 Information Processing Systems - Open Systems Interconnection - Specification of the Basic Class Protocol for Job Transfer and Manipulation.

*ISO DIS 9040/1 Information Processing Systems - Open Systems Interconnection - Virtual Terminal Service - Basic Class - Part 1: Initial Facility Set

*ISO DIS 9041/1 Information Processing Systems - Open Systems Interconnection - Virtual Terminal Protocol - Basic Class - Part 1: Initial Facility Set

*ISO DIS 8649/2 Information Processing Systems - Open Systems Interconnection - Definition of Common Application Service Elements - Part 2: Association Control.

*ISO DIS 8649/3 Information Processing Systems - Open Systems Interconnection - Definition of Common Application Service Elements - Part 3: Commitment Concurrency and Recovery

Presentation Layer Standards

*ISO DIS 8822 Information Processing Systems - Open Systems Interconnection - Connection Oriented Presentation Service Definition

*ISO DIS 8823 Information Processing Systems - Open Systems Interconnection - Connection Oriented Presentation Protocol Specification

*ISO DIS 8824 Information Processing Systems - Open Systems Interconnection - Abstract Syntax Notation 1.

*ISO DIS 8825 Information Processing Systems - Open Systems Interconnection - Basic encoding rules for Abstract Syntax Notation 1 (ASN.1)

Session Layer Standards

ISO 8326 Information Processing Systems - Open Systems Interconnection - The Basic Connection Oriented Session Service Definition.

Registration Authorities

ISO TC 97/SC 21 N 436 Procedures for OSI Registration Authority

ISO TC 97/SC 21 N 511 Registration of Upper Layer Syntaxes

ISO TC 97/SC 21 N 969 Procedures for Registration Authority for Document Types

Formal Description Techniques

*ISO DP 9074 Information Processing Systems - Open Systems Interconnection - ESTELLE - A Formal Description Technique Based on an Extended State Transition Model

*ISO DP 8807 Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour

ISO TC 97/SC 21 N 933 Guidelines for the application of formal description techniques to OSI

ISO TC 97/SC 21 N 937 Provisional Estelle tutorial

ISO TC 97/SC 21 N 937 Provisional LOTOS tutorial

Conformance Testing

ISO TC 97/SC 21 N 410R Revised draft for OSI Conformance testing methodology and framework

ISO TC 97/SC 21 N 935 Development and acceptance procedures for formally described OSI protocols and services

Banking Standards

ISO DIS 8583 Bank Card Originated Messages - Interchange Message Specifications - Content for Financial Transactions