A SESSION LAYER FOR THE X.400 MESSAGE HANDLING SYSTEM

by

EUGENE DANIEL VAN DER WESTHUIZEN

Submitted in fulfilment of the requirements for

the degree of

Master of Science

in the

Department of Electrical and Electronic Engineering

University of Cape Town

February 1990

Preface

The work for, and preparation of, this thesis were done while the author was a full time student in the Department of Electrical and Electronic Engineering at the University of Cape Town from March 1987 to January 1990. Supervision was by Mr. M.J.E. Ventura.

These studies represent original work by the author and have not been submitted in any other form to another university. Where use was made of the work of others it has been duly acknowledged in the text.

Signed: .....

E.D. van der Westhuizen

Department of Electrical and Electronic Engineering
University of Cape Town

Acknowledgments

The author would like to thank the following persons and institutions:

Mr. M.J.E. Ventura of the Department of Electrical and Electronic Engineering, my supervisor, for his help and guidance with this thesis, and for reading the final script.

Mr. Graham Jack of the Department of Electrical and Electronic Engineering for invaluable, practical help with any conceivable computer-related issues.

The South African Council for Scientific and Industrial Research for financial support for this thesis in the form of a post graduate bursary.

Abstract

The CCITT X.400 Message Handling System resides in the Application Layer of the seven-layer Reference Model for Open Systems Interconnection. It bypasses the services of the Presentation Layer completely to interact directly with the Session Layer.

The objectives of this thesis are to show how the general Session Layer may be tailored to be minimally conformant to the requirements of X.400; to produce a formal specification of this session layer; and to show how this session layer may be implemented on a real system.

The session services required by X.400 are those of the Half-duplex, Minor Synchronization, Exceptions and Activity Management functional units of the CCITT X.215 Session Service Definition. These services, and particularly their use by X.400, are described in detail. State tables describing these services are derived from the general session service state tables.

Those elements of the CCITT X.225 Session Protocol Specification which are required to provide only those services required by X.400 are described in detail. State tables describing this session protocol are derived from the general session protocol state tables.

A formal specification of the session layer for X.400 is presented using the Formal Description Technique Estelle. This specification includes a complete session entity, which characterizes the entire session layer for X.400.

A session entity for supporting X.400 is partially implemented and interfaced to an existing X.400 product on a real system. Only the Session Connection Establishment Phase of the session protocol is implemented to illustrate the technique whereby the entire session protocol may be implemented. This implementation uses the C programming language in the UNIX operating system environment.

TABLE OF CONTENTS

List of figures

List of tables

List of acronyms

APDU - Application Protocol Data Unit
CCITT - The International Telegraph and Telephone Consultative Committee
CEP - Connection End Point
ECMA - European Computer Manufacturers' Association
FDT - Formal Description Technique
FIFO - First In First Out
ISO - The International Standards Organization
MH - Message Handling
MHS - Message Handling System
MTA - Message Transfer Agent
OSI - Open Systems Interconnection
QOSS - Quality Of Session Service
QOTS - Quality Of Transport Service
RTS - Reliable Transfer Server
SAP - Service Access Point
SC - Session Connection
SCEP - Session Connection End Point
SIDU - Session Interface Data Unit
SPDU - Session Protocol Data Unit
SPM - Session Protocol Machine
SS - Session Service
SSAP - Session Service Access Point
SSDU - Session Service Data Unit
TC - Transport Connection
TCEP - Transport Connection End Point
TIDU - Transport Interface Data Unit
TPDU - Transport Protocol Data Unit
TS - Transport Service
TSAP - Transport Service Access Point
TSDU - Transport Service Data Unit
UA - User Agent

1. INTRODUCTION

The CCITT has recently (1985) defined a new, generic, global telecommunications service known as Message Handling. This service combines computer-based electronic mail systems with established telematic services such as telex and facsimile. This service is defined in the CCITT X.400 series of Message Handling System (MHS) Recommendations [1]. The entities providing this service reside in the application layer (layer seven) of the Reference Model of Open Systems Interconnection (OSI) [2] and make extensive use of the lower layers of the Model.

Of particular interest to X.400 is the session layer, the fifth layer of the OSI Reference Model. This layer provides dialogue control and management services to application entities through layer six, the presentation layer. However, the X.400 lower-layer service requirements have been designed to by-pass the presentation layer altogether, so that the X.400 application entities interact directly with the session layer. The general session layer used by X.400 is defined by the Session Service Definition of CCITT Recommendation X.215 [3] and the Session Protocol Specification of CCITT Recommendation X.225 [4].

The general session layer is a large and complex layer providing many optional services to meet the requirements of any type of application. X.400, however, uses only a subset of all possible session services, and may therefore be supported by a tailored version of the general session layer which provides only those services it needs. Describing and implementing such a session layer for X.400 is the subject of this thesis.

The objectives of this thesis are:

1. to show how the general session layer may be tailored to meet only the requirements of X.400;

2. to present a formal description of this session layer;

3. to show how this session layer may be implemented and interfaced to an existing X.400 application on a real system.

The reader is assumed to be thoroughly familiar with the concepts and terminology of the OSI Reference Model [2] and the OSI Layer Service Definition Conventions [5]. In addition, a thorough knowledge of the ISO Formal Description Technique *Estelle* [6], the C Programming Language [7] and the UNIX Operating System [8] is assumed when the formal description and implementation of the session layer is presented.

The material presented in the rest of this thesis is organized as follows:

Section 2 presents an overview of the general session layer. It briefly reviews all relevant concepts and terminology of the OSI Reference Model and OSI Layer Service Definition Conventions. It then broadly describes the purpose of the session layer, its use and provision of layer services, and elements of its internal operation.

Section 3 presents an overview of the X.400 Message Handling System. It briefly describes its architecture and shows why the session layer is of special importance to it.

Section 4 presents a detailed definition of the session service which is minimally conformant to the requirements of X.400 only. This is derived from the general Session Service Definition of CCITT Recommendation X.215.

This information is required by section 5, which presents a detailed specification of the session protocol required to provide only those session services required by X.400. This is derived from the general Session Protocol Specification of CCITT Recommendation X.225.

These informal descriptions of the session service and protocol required by X.400 are combined by section 6 into a complete, formal description of the session layer for X.400, using the ISO Formal Description Technique, Estelle.

Based on this formal description, section 7 shows how this session layer may be implemented on a real system. It presents a partial implementation of this session layer, written in the C programming language and interfaced to an existing X.400 application in the UNIX operating system environment.

Finally, section 8 concludes whether this description and implementation of the session layer for X.400 meets the objectives of this thesis.

2. OVERVIEW OF THE SESSION LAYER

This section presents a general overview of the session layer. First, it shows how the session layer fits into the OSI environment, identifying and reviewing relevant concepts and terminology of the OSI Reference Model and the OSI Layer Service Definition Conventions. These concepts are not thoroughly defined here, as such definitions may be found in CCITT Recommendations X.200 [2] and X.210 [5]. This is followed by a description of the purpose of the session layer. Broad descriptions are then given of the services available to the session layer from the transport layer, the services provided by the session layer to its users, and the major, internal session layer functions.

2.1 The session layer in the OSI environment

Figure 2.1 depicts the session layer in the OSI environment, showing elements of its internal structure and interaction with adjacent layers. This is followed by a brief description of each element.

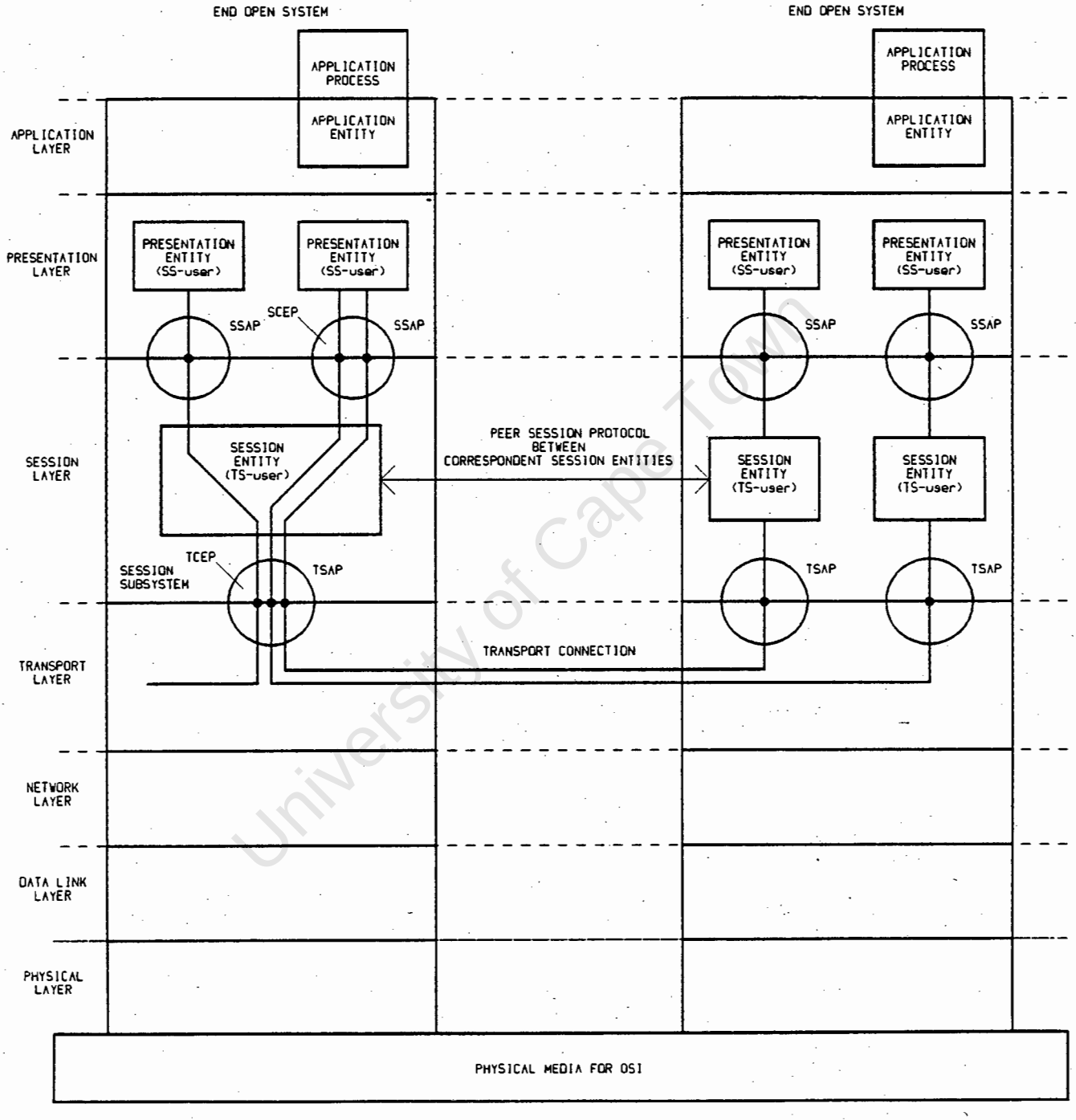

Figure 2.1 The session layer in the OSI environment

Layering:

Open systems are real systems which employ the standardized communication procedures derived from the OSI Reference Model.

An *application process* performs information processing for an application, while an *application entity* represents those aspects of the application process of concern to OSI.

A *session subsystem* is that element of the hierarchical division of an open system representing session layer functionality. The session subsystems in all open systems collectively form the *session layer*. A session subsystem consists of one or more active elements called *session entities*. All session entities within the session layer are called *peer* session entities.

A *service* is a capability provided by one layer to the layer above it. Session entities are responsible for providing *session services* directly to presentation entities. A *Session Service Access Point* (SSAP) is the point at which one session entity provides session services to one presentation entity. In providing the session service, session entities communicate with each other using the (peer) *session protocol* via services provided by the transport layer.

A *Transport Service Access Point* (TSAP) is the point at which one session entity uses transport services provided by one transport entity.

Communication between peer entities:

A *session connection* is an association for communication established by the session layer between two correspondent presentation entities. Session connections are provided between two SSAPs, where they are terminated by *Session Connection Endpoints* (SCEPs). Similarly, correspondent session entities communicate via a *transport connection* accessible at *Transport Connection End Points* (TCEPs) within their TSAPs.

Identifiers:

A presentation entity is uniquely identified by its SSAP *address*, or *session address*. Similarly, a session entity is uniquely identified by its TSAP *address*, or *transport address*. Connection endpoints within a SAP are identified by *Connection Endpoint identifiers*, which are unique within the scope of the entity supported by the SAP.

Addressing:

In the application layer, applications are referenced by a *directory* function which maps *application titles* into the PSAP addresses through which they may be accessed. Below the transport layer, a directory function provides the mapping between an NSAP address and the routing information required to create a path to the destination NSAP.

For the middle layers (presentation, session and transport), a unique (N)-address consists of its unique, supporting (N-1)-address plus an *(N)-suffix* which is unique within the scope of the (N-1)-address. This leads to a simple *hierarchical* address arrangement. An (N)-suffix is also called an *(N)-SAP selector* or an *(N)-SAP identifier*. The use of selectors is not mandatory, allowing *one-to-one mappings* between (N) and (N-1) addresses. The address information for a given layer (the (N)-

suffix, selector or identifier) is always conveyed within the protocol of that layer.

Using this addressing scheme, the address of an application entity can therefore only ever comprise:

Application address = NSAP address + TSAP selector
+ SSAP selector
+ PSAP selector.

Data units:

Session Protocol Control Information (SPCI) is data passed between session entities to coordinate their joint operation. *Session User Data* (SUD) is passed between session entities on behalf of presentation entities. A *Session Protocol Data Unit* (SPDU) is a unit of data specified in the session protocol and consists of SPCI and possible SUD.

Session Interface Control Information (SICI) is passed between presentation and session entities to coordinate their joint operation. *Session Interface Data* (SID) is data passed from a presentation entity to a session entity for transmission to a remote presentation entity, or data passed from a session entity to a presentation entity after being received from a remote presentation entity. A *Session Service Data Unit* (SSDU) is a unit of data passed between two presentation entities whose integrity is to be maintained end-to-end. A *Session Interface Data Unit* (SIDU) is the unit of data passed across an SSAP in a single interaction and consists of SICI and possibly SID, which is the whole or part of an SSDU. Similar data units are defined for interactions between session and transport entities across a TSAP. They are:

Transport Interface Control Information (TICI),
Transport Interface Data (TID),
Transport Service Data Unit (TSDU),
Transport Interface Data Unit (TIDU).

Layer services:

A *service user* represents all those entities in an open system that make use of a service through a SAP. A presentation entity is therefore a session service user (SS-user) which uses session services through an SSAP. Similarly, a session entity is a transport service user (TS-user) which uses transport services through a TSAP.

A *service provider* is an abstract machine which models the behaviour of all those entities providing the service, as viewed by the user. SS-users therefore communicate by means of the session service provider (SS-provider) and TS-users communicate by means of the transport service provider (TS-provider).

Each service user interacts with the service provider by issuing or receiving *service primitives* at a SAP. The four types of service primitive are:

request (from user to provider),
indication (from provider to user),
response (from user to provider),
confirm (from provider to user).

2.2 The purpose of the session layer

The session layer is the fifth layer of the seven-layer OSI Reference Model. Broadly, its purpose is to add application-orientated services to the end-to-end communications channels provided by the transport layer. More specifically, it provides the means necessary for cooperating SS-users to organize and synchronize their dialogue and to manage their data exchange. To do this, the session layer provides services to:

a) establish a session connection between two SS-users;

b) support orderly data exchange interactions during the lifetime of the session connection; and

c) release the session connection.

The session service is provided by the session protocol using services available from the transport layer.

2.3 Services available from the transport layer

The transport layer provides TS-users in end open systems with a network-independent, transparent, data transfer service. It shields the higher layers from the technical details of how the communication is achieved. It optimizes the use of available network resources while maintaining, at minimum cost, a guaranteed quality of service required by the session entities.

2.3.1 Transport connection establishment

This service enables two TS-users to establish a transport connection between themselves. A transport connection is simply a full-duplex data path. The two session entities are identified by their unique TSAP addresses.

A TS-user may be associated with several transport connections simultaneously, to either the same or different TS-users. Both concurrent and consecutive transport connections are possible between two TS-users. Multi-endpoint transport connections are not allowed.

Each TS-user is provided with a TCEP identifier, enabling it to distinguish the new transport connection from all others accessible at its TSAP.

The quality of service required by the TS-users on the transport connection is negotiated between the TS-users and the TS-provider.

2.3.2 Normal data transfer

This service provides a reliable, full-duplex, transparent transfer of normal TSDUs over a transport connection, preserving TSDU boundaries, contents and sequence.

The agreed quality of service must be maintained by the TS-provider while the transport connection exists. If the TS-provider can no longer maintain that quality, it terminates the transport connection and notifies the TS-users of this fact.

This service is also subject to flow control, allowing receiving TS-users to control the rate at which sending TS-users may send data. The decision on when or how this flow control is applied is a local matter and is therefore not subject to any OSI specification.

2.3.3 Expedited data transfer

This service allows the transfer of small, expedited TSDUs over a transport connection. These TSDUs are transferred and/or processed with priority over normal TSDUs. This service is intended for signalling and interrupt purposes and may be used by either TS-user at any time that a transport connection exists.

2.3.4 Transport connection release

This service allows either TS-user to unconditionally release a transport connection and have the correspondent TS-user informed of the release.

2.4 Services provided by the session layer

2.4.1 Session connection establishment

This service enables two SS-users to establish a session connection, or *session*, between themselves, a process often called *binding*. The SS-user entities are identified by their unique SSAP addresses. The SS-users may negotiate and agree on a variety of options and parameters that may or may not be in effect for the session connection.

An SS-user may be associated with several session connections simultaneously, to either the same or different SS-users. Both concurrent and consecutive session connections are possible between two SS-users. Simultaneous session connection establishment requests may result in a corresponding number of session connections, but a session entity can always reject an incoming request. Multi-endpoint session connections are not allowed.

Each SS-user is provided with a SCEP identifier, enabling it to distinguish the new session connection from all others accessible at its SSAP.

2.4.2 Normal data transfer

This service allows a sending SS-user to transfer a normal SSDU to a receiving SS-user. This service also allows the receiving SS-user to ensure that it is not overloaded with data.

2.4.3 Expedited data transfer

This service allows the transfer of small, expedited SSDUs between SS-users. These SSDUs are transferred and/or processed with priority over normal SSDUs. This service is intended for signalling and interrupt purposes, and may be used by either SS-user at any time that a session connection exists.

2.4.4 Interaction management

This service allows the SS-users to control explicitly whose turn it is to exercise certain control functions. This service provides for:

a) *voluntary* exchange of the turn, where the SS-user which has the turn relinquishes it voluntarily, and

b) *forced* exchange of the turn, where, upon request from the SS-user which does not have the turn, the session service may force the SS-user with the turn to relinquish it. In this case, data may be lost.

This service enables SSDU exchange interactions to be either full-duplex (two-way simultaneous), half-duplex (two-way alternate) or simplex (one-way).

2.4.5 Session connection synchronization

This service allows SS-users to establish synchronization points in their dialogue. Once a synchronization point has been established and confirmed by the SS-users, both view the data transferred prior to the synchronization point as secured.

In the event of errors, this service then allows the SS-users to reset the session connection to a defined state and resume the dialogue from an agreed resynchronization point. The session layer is not responsible for any associated checkpointing or commitment action associated with resynchronization.

This service aids SS-users in recovering from communication failure without losing all the data already transferred - only unconfirmed data need be retransmitted.

2.4.6 Exception reporting

This service allows the SS-users to be notified of exceptional circumstances not covered by other services, such as unrecoverable SS-provider malfunctions or SS-user errors. It should be noted that the type of errors encountered in the session layer are procedural errors or failures of the underlying transport connection. Actual transmission errors are dealt with in the lower layers of the OSI Reference Model.

2.4.7 Session connection release

The session connection exists until released by either the SS-users or the session entities. This service allows SS-users to release a session connection in an *orderly* way without loss of data. It also allows either SS-user to request at any time that a session connection be *aborted*. In this case, data may be lost. The release of a session connection may also be initiated by one of the session entities supporting it.

2.5 Functions within the session layer

The functions within the session layer are those which must be performed by session entities in order to provide the session services. These functions are visible only within the session protocol and are therefore transparent to the presentation and transport layers. The major functions are described below:

2.5.1 Session address mapping

Generally, there is a many-to-one, hierarchical mapping between session and transport addresses. This does not imply multiplexing of session connections onto transport connections, but does imply that at session connection establishment time, more than one SS-user is a potential target of a session connection establishment request arriving on a given transport connection. The target SS-user is identified by the SSAP selector carried by the session protocol.

In many systems, a transport address may be used as the session address, i.e., there is a one-to-one mapping between the session and transport addresses.

2.5.2 Session connection mapping

To implement the transfer of data between the SS-users, the session connection is mapped onto, and uses, a transport connection. If a suitable transport connection is not available at session connection establishment time, one must be established. There is always a one-to-one mapping between session and transport connections. Conversely, there is no multiplexing or splitting between session and transport connections. A transport connection may, however, support several consecutive session connections.

To implement the mapping of a session connection onto a transport connection, the session layer must map SSDUs into SPDUs (possibly using the complementary operations of segmenting and reassembly), and SPDUs into TSDUs (possibly using concatenation and separation). SSDUs may not be mapped into SPDUs using blocking and deblocking.

2.5.3 Flow control

There is no peer flow control in the session layer. To prevent the receiving SS-user from being overloaded with data, it applies back pressure across the transport connection by using the transport flow control. However, this is a local matter and is therefore not subject to any OSI specification.

2.5.4 Expedited data transfer

The transfer of expedited SSDUs is generally accomplished by use of the transport expedited data service.

3. OVERVIEW OF THE X.400 MESSAGE HANDLING SYSTEM

The CCITT X.400 Message Handling System (MHS) is a generic, global, medium-independent, store-and-forward, electronic mail service. It standardizes electronic mail systems by integrating computer-based message systems with established, heterogeneous telematic services such as telex, teletex, facsimile, videotex, voice, etc. It allows its users to communicate by exchanging *messages*. Messages are comprised of addressing information and user content, which may include any combination of text, facsimile, graphics or other data structures.

The MHS is defined in CCITT Recommendations X.400 to X.430 [1] as a set of standard protocols, service definitions and user interfaces. These services and protocols reside in the application layer of the OSI Reference Model [2] and make extensive use of the Model's lower layers.

This section presents a general overview of the MHS, briefly describing its architecture and identifying its requirements of the lower layers of the OSI Reference Model. In particular, it shows why the session layer is of special interest to the MHS.

3.1 A functional model of the MHS

Figure 3.1 depicts a functional model of the MHS. This is followed by a brief description of the model's components and their functions.

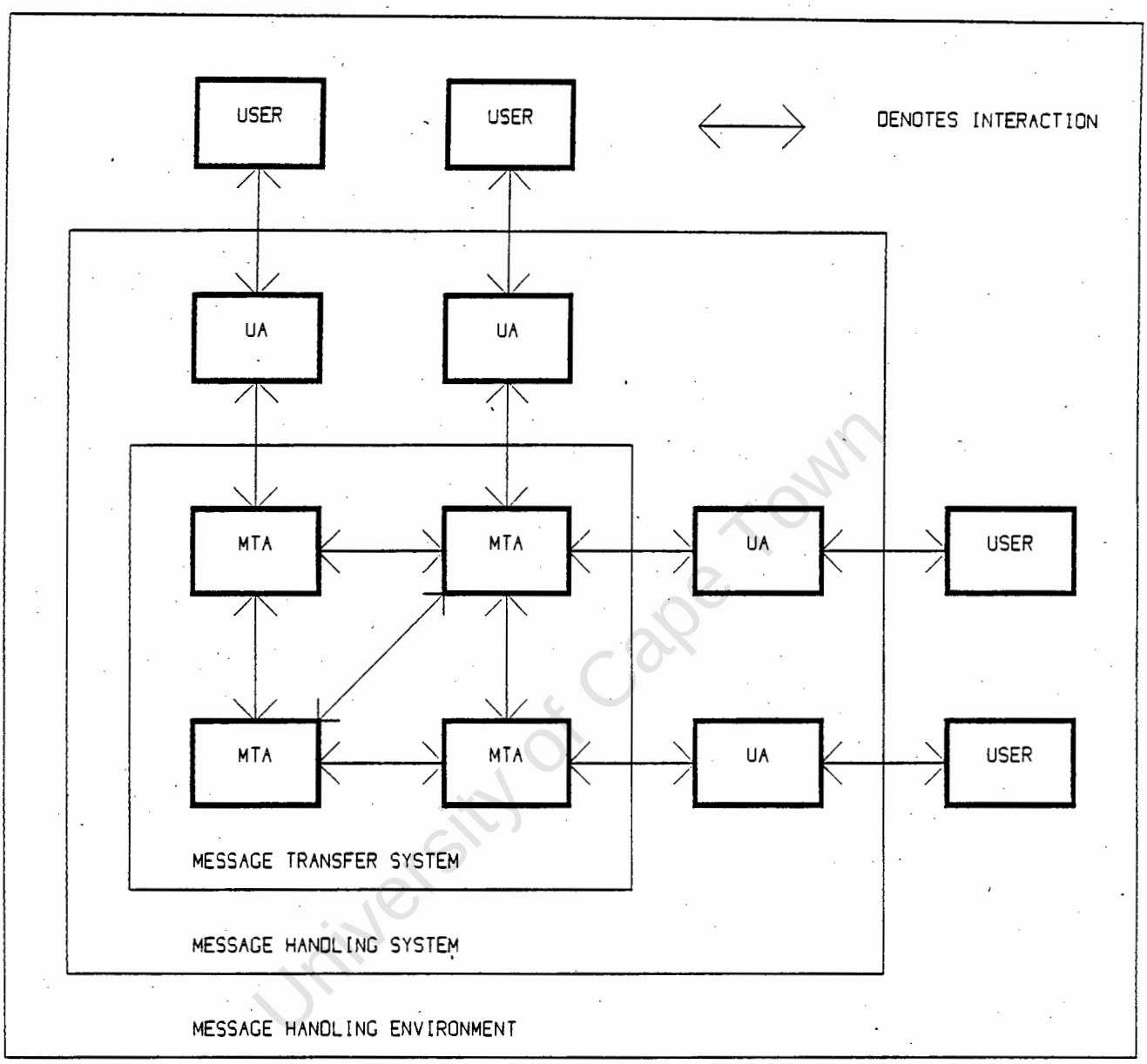

Figure 3.1 A functional model of the MHS

A *user* is either a person or a computer application. It is referred to as an *originator* (when sending a message) or a *recipient* (when receiving a message). An originator prepares a message with the assistance of its *User Agent* (UA). A UA is an application process that interacts with the *Message Transfer System* (MTS) to submit messages. The MTS delivers to one or more recipient UAs the messages submitted to it.

The MTS comprises a number of *Message Transfer Agents* (MTAs). Operating together, the MTAs relay messages and deliver them to the intended recipient UAs, which make the messages available to the intended recipients.

The collection of UAs and MTAs is called the *Message Handling System* (MHS). The MHS and all of its users are collectively referred to as the *Message Handling Environment*.

Messages consist of an *envelope* and *content*. The envelope carries addressing information used when transferring the message, while the content is the piece of information that the UA wishes delivered to one or more recipient UAs.

3.2 A layered model of the MHS

Figure 3.2 depicts a layered model of the MHS, showing how it fits into the OSI environment. This is followed by a brief description of the model's features.

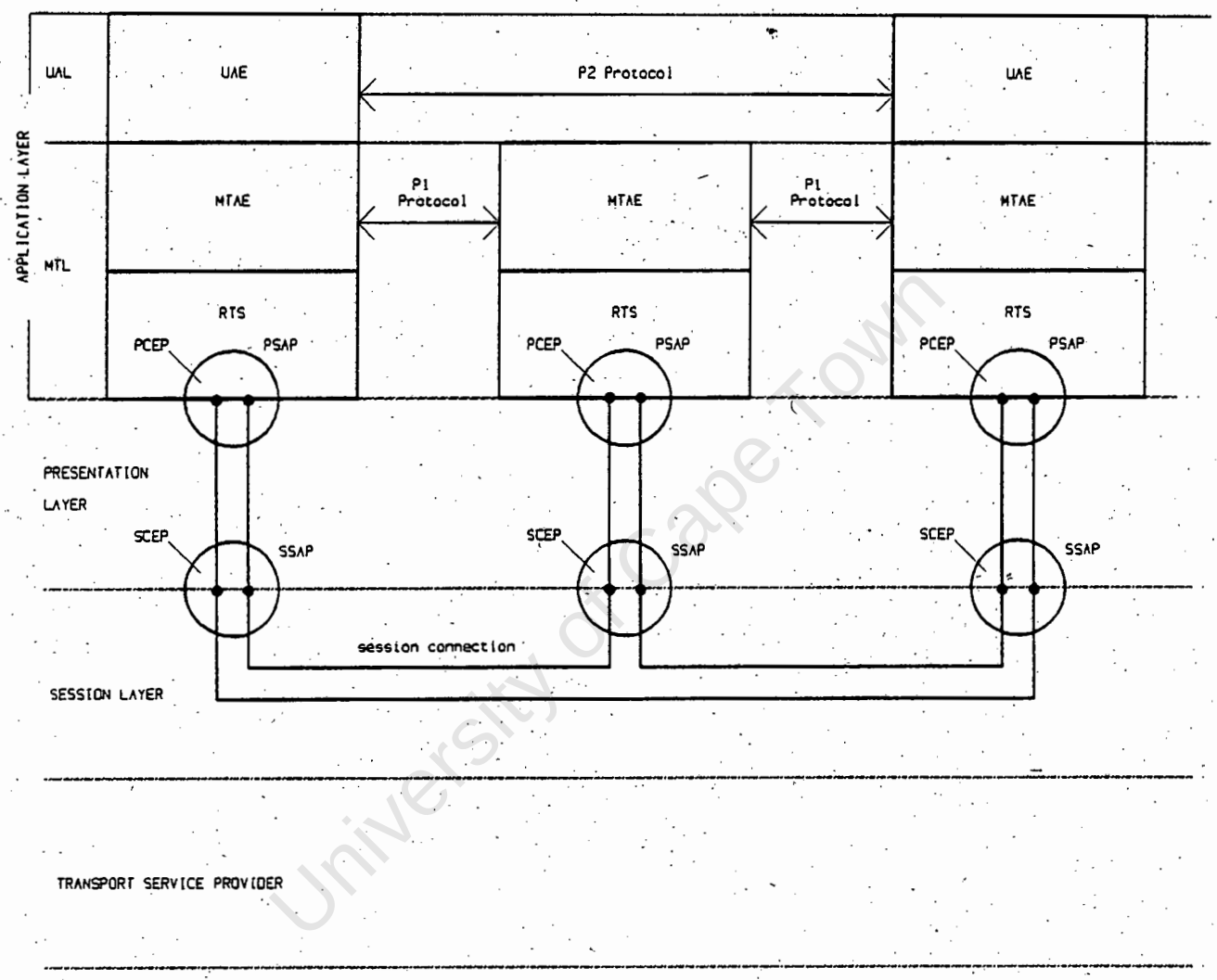

Figure 3.2 A layered model of the MHS

The MHS entities and protocols reside in the application layer of the OSI Reference Model. This allows the MHS application to use the underlying layers to establish network-independent connections between individual systems, and to establish session connections, permitting the MHS applications to reliably transfer messages between open systems.

The CCITT has split the application layer into two sublayers for the MHS application:

a) the *User Agent Layer* (UAL) contains the functionality associated with the contents of messages and is driven by the users;

b) the *Message Transfer Layer* (MTL) contains the MTA functionality and provides the MTS to the UAL.

The UA entity (UAE) embodies only those aspects of UA functionality associated with the operation of the UA to UA protocol (P2), not the local UA functionality. The MTA entity (MTAE) provides the functionality required to support the layer services of the MTL in cooperation with other MTAEs. The message transfer protocol (P1) is responsible for relaying messages between MTAEs and other interactions necessary to provide the MTL services.

A MTAE is divided into two elements: the *Message Dispatcher* and the *Reliable Transfer Server* (RTS). The Message Dispatcher performs the P1 protocol, which relies on the lower-level OSI protocols through the RTS. The RTS is therefore responsible for creating and maintaining connections between the MTAE and its peers, and for reliably transferring P1 Application Protocol Data Units (APDUs) by means of them. The RTS is therefore the subsystem in the MHS application layer that interfaces with the lower layers of the OSI Model. The detailed operation of the RTS and its use of the lower layers is described in CCITT Recommendation X.410 [9].

As will be shown in section 3.3.1, the RTS makes direct use of the session layer services. Since the OSI Reference Model permits direct interactions only between adjacent layers, the RTS cannot (strictly speaking) interact directly with session entities. This interaction is thus described as "through" the presentation layer which intervenes "transparently" to convey the interactions between the RTS and the session layer. This scheme is illustrated in Figure 3.2 as one-to-one mappings between PSAPs and SSAPs, and presentation "connections" and session connections. These mappings are consistent with the description of the presentation layer in the OSI Reference Model.

3.3 Use of layers below the application layer

The MHS APDUs are communicated between end systems by the operation of protocols in the lower six layers of the OSI Reference Model. Of particular interest to the MHS are the presentation and session layers.

3.3.1 The presentation layer

In general, the presentation layer serves to determine a common transfer syntax for the exchange of APDUs between application entities. Where the syntax preferred or used by each application entity differs, the presentation layer would provide a translation or mapping of one syntax to another, or each into a common transfer syntax.

This function of syntax conversion is particularly important in MH, as one of the features of MH is its automatic conversion of user data of one kind (e.g. text), to data of another kind (e.g. facsimile). A presentation protocol could be used to determine conversion requirements, and relieve the application protocol of these considerations.

However, the development of suitable protocols and conversion algorithms for a general presentation layer have lagged the work - and requirements - of MH by some years. In order to achieve functioning MH protocols in time for the 1984 CCITT Plenary, MH was designed to bypass entirely the presentation layer. The presentation layer is being defined to permit this type of approach, in which application entities can access the session services directly, so MH remains consistent with the OSI Reference Model.

3.3.2 The session layer

In the absence of a presentation layer protocol in MH systems, the session layer is directly responsible for the secure delivery of MH APDUs. This means that the Reliable Transfer Server (RTS) actually interacts directly with the session layer.

MH uses the connection-orientated session service defined in CCITT Recommendation X.215 [3], which employs the session protocol specified in CCITT Recommendation X.225 [4]. The detailed use of the session service by MH is the subject of the next section, section 4.

3.3.3 The transport layer and below

MH uses the connection-orientated transport service defined in CCITT Recommendation X.214 [10], which employs the transport protocol specified in CCITT Recommendation X.224 [11].

Protocols below the transport layer are network-dependent and are therefore not specified for MH. MH systems can thus be implemented over any telecommunications network providing adequate quality of service. In practice, MH

would often be implemented on a packet-switched network with CCITT X.25 the appropriate protocol.

4. THE SESSION SERVICE FOR X.400

The session service defines in an abstract, conceptual, implementation-independent way the essential properties and features of the service provided by the session layer to its users. It is defined in terms of:

a) the primitive events and actions of the service;
b) the parameter data associated with each primitive action and event;
c) the relationships between, and the valid sequences of the actions and events.

The general session layer provides a great many optional services to SS-users. This enables it to support all possible application types. However, the CCITT X.400 application represents only a subset of all possible application types, and therefore requires only a subset of the general session services.

Identifying this subset of session services required by X.400 is the subject of this section. It specifies precisely which of the general session services are used by X.400 and how X.400 uses them. This information is required by section 5, which will show how the general session protocol may be tailored to provide only these services required by X.400.

Section 3 identified the RTS as that element of an X.400 application entity which interfaces directly with the session layer. The RTS is therefore an X.400 SS-user. This section derives the session service for the RTS by applying the RTS's session service requirements, as specified in CCITT Recommendation X.410 section 4 [9], to the general Session Service Definition of CCITT Recommendation X.215 [3].

4.1 Definition of terms

For the purpose of this section and the rest of this thesis, the following definitions apply:

calling SS-user
An SS-user that initiates a session connection establishment request.

called SS-user
An SS-user with whom a calling SS-user wishes to establish a session connection.

Note: Calling SS-users and called SS-users are defined with respect to a single connection. An SS-user can be both a calling and a called SS-user simultaneously.

sending SS-user
An SS-user that sends data during the data transfer phase of a session connection.

receiving SS-user
An SS-user that receives data during the data transfer phase of a session connection.

requestor; requesting SS-user
An SS-user that initiates a particular action.

acceptor; accepting SS-user
An SS-user that accepts a particular action.

conditional parameter
A parameter whose presence in a request or response primitive depends on some condition; and whose presence in an indication or confirm primitive is mandatory if it was present in the preceding primitive, or absent if it was absent in the preceding primitive.

proposed parameter
The value for a parameter proposed by an SS-user, in a session connection request or response primitive, that it wishes to use on the session connection.

selected parameter
The value for a parameter that has been chosen for use on the session connection.

4.2 Model of the session service

Figure 4.1 depicts a model of the session service. It shows the SS-provider providing a session connection between two correspondent SS-users. Each SS-user accesses session services at its SSAP, in which the session connection is terminated by a SCEP.

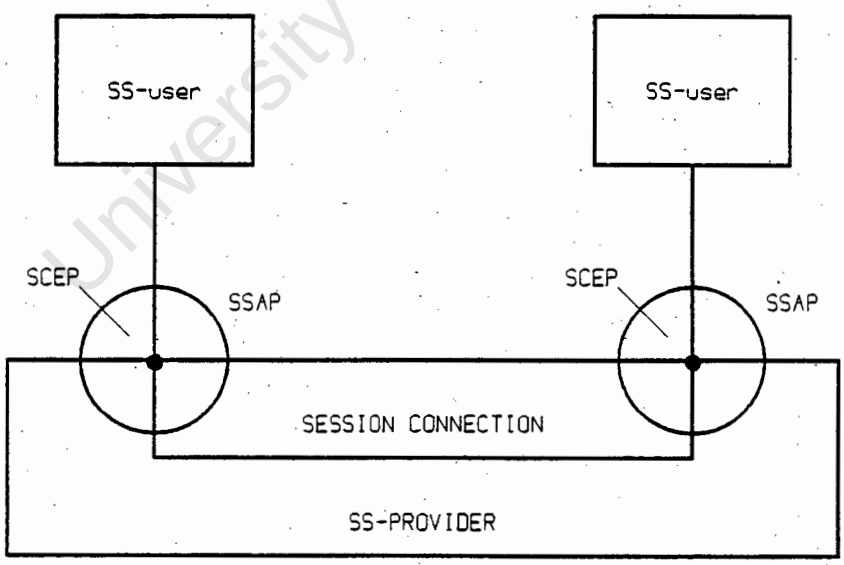

Figure 4.1 Model of the session service

4.3 The token concept

A token is an attribute of a session connection which is dynamically assigned to one SS-user at a time to permit certain services to be invoked. It controls the right to exclusive use of the service.

Four tokens are defined, each controlling the use of one or more services. Table 4.1 lists the tokens, their standard abbreviated names and the services controlled by each.

Table 4.1 Tokens

token	abbrv	services controlled
data	dk	normal data transfer
release	tr	orderly connection release
synchronize-minor	mi	minor synchronization
major/activity	ma	major synchronization activity management

A token is always in one of the following states:

a) *available*, in which case it is always:

   1) *assigned* to one SS-user (the *owner* of the token), who then has the exclusive right to use the associated service (provided that no other restrictions apply);

   2) *not assigned* to the other SS-user, who does not have the right to use the service but may acquire it later;

b) *not available* to either SS-user, in which case neither SS-user has the exclusive right to use the associated service. The service then becomes inherently available to both SS-users (data transfer and orderly release), or unavailable to both SS-users (synchronization and activity management).

The RTS restricts its use of the tokens as follows:

a) The data, synchronize-minor and major/activity tokens are always available. The release token is never available.

b) The tokens are never separated, i.e., all the available tokens are always assigned to one of the correspondent RTSs. The owner of the tokens is referred to as the *sending* RTS, the other as the *receiving* RTS.

Subsection 4.8.1 shows how specific tokens are made available to a session connection, while 4.10.2 defines the token restrictions placed on session services.

4.4 The major synchronization and activity concepts

Certain session services allow the SS-users to partition parts of their dialogue. The SS-users may 'mark' points in their dialogue, upon which the session layer ensures complete separation of the dialogue before and after the mark. This can be done by using either "Major Synchronization Points" or "Activities".

These two styles for SS-user dialogue separation, and corresponding resynchronization procedures, stem from the fact that a major concern during the development of the session layer standards has been the issue of compatibility with existing CCITT standards. As a result, "Activities" are intended for use by CCITT applications, while "Major

Synchronization Points" are intended for ECMA applications. Since X.400 is a CCITT application, the RTS uses the activity concept for separating its dialogues.

A brief description of both these concepts is presented below, showing why the activity concept is better suited to RTS requirements than the major synchronization concept.

4.4.1 Major synchronization

Major synchronization points are intended for separating general session dialogues which use full-duplex normal and expedited data exchange.

SS-users may insert major synchronization points into the data they are transmitting, each identified by a serial number maintained by the SS-provider. Any semantics which SS-users may give to their major synchronization points are transparent to the SS-provider.

Major synchronization points structure the data exchange into a series of *dialogue units*. The characteristic of a dialogue unit is that all communication within it is completely separated from all communication before and after it. A major synchronization point indicates the end of one dialogue unit and the start of the next. Each major synchronization point must be confirmed explicitly.

An example of using major synchronization points would be to indicate a change in application dialogue context, e.g. from file transfer to message-passing.

Major synchronization is not suited to RTS requirements because, as will be shown, the RTS never uses full-duplex or expedited data exchanges. In addition, RTS dialogue context remains constant - that of Message Handling.

4.4.2 Activities

Activities are intended for separating half-duplex data exchanges.

The activity concept allows SS-users to distinguish between different logical pieces of work called *activities*. Each activity may be structured into one or more dialogue units by use of major synchronization points. Only one activity is allowed on a session connection at a time, but there may be several consecutive activities during a session connection. An activity can be interrupted and then resumed on the same or on a subsequent session connection. This can be considered as a form resynchronization. The SS-users may transfer only *capability* data outside an activity.

An example of using activities would be the separation of documents on a document transfer connection.

Activities are suited to RTS requirements because RTS dialogue is always half-duplex.

4.5 The minor synchronization point concept

Minor synchronization points are intended for partitioning simplex data flow with no use of session expedited data.

SS-users may insert minor synchronization points into the data they are transmitting. Each minor synchronization point is identified by a serial number maintained by the SS-provider. Any semantics which SS-users may give to their minor synchronization points are transparent to the SS-provider. Each minor synchronization point may or may not be explicitly confirmed.

Minor synchronization points are used to structure data within either dialogue units or activities. Figure 4.2 shows how an RTS activity may be structured through the use of minor synchronization points.

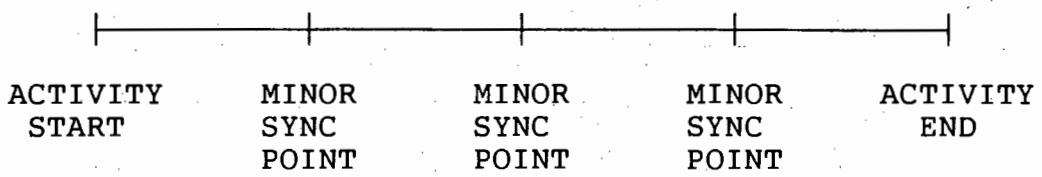

Figure 4.2 Example of a structured activity

4.6 The resynchronization concept

Resynchronization may be initiated by either SS-user. It sets the session connection to a defined state, reassigns the tokens, sets the synchronization point serial number to a new value and purges all undelivered data. Resynchronization is never used by the RTS. Instead, the RTS uses a form of resynchronization associated with activities.

4.7 Phases and services of the general session service

The general session service comprises three phases, each of which provides certain services. The purpose of each phase and a brief description of the associated services is given here. Of these services, those used by the RTS are identified.

4.7.1 The session connection establishment phase

This phase is concerned with establishing a session connection between two SS-users. It has one service associated with it:

a) The **Session Connection** service.
This service is always provided to all SS-users. It enables two SS-users to establish a session connection between themselves. Simultaneous attempts by both SS-users to establish a connection may result in two session connections. An SS-user may always reject an unwanted connection. No architectural restriction is placed on the number of concurrent session connections associated with an SS-user, or between two SS-users.

The SS-users may negotiate the values of various session connection parameters. By the end of the session connection establishment phase, the SS-users have agreed on a set of parameter values concerning the session connection.

4.7.2 The data transfer phase

This phase is concerned with the exchange of data between the two SS-users connected in the session connection establishment phase.

There are four services concerned with data transfer:

a) The **Normal Data Transfer** service.
This service is always provided on every session connection. It allows SS-users to exchange unconfirmed, unlimited-length, normal SSDUs over a session connection. The SS-provider delivers each normal SSDU to the SS-user as soon as possible. Use of this service is controlled by the data token if it is available.

b) The **Expedited Data Transfer** service.
This optional service allows SS-users to exchange unconfirmed, limited-length, expedited SSDUs over a session connection, free from the token and flow control constraints of the other three data transfer services. This service is not used by the RTS.

c) The **Typed Data Transfer** service.
Generally, SS-user data consists of two distinct types: application user data and PDUs from Layers 6 and 7. For some applications, the latter should not be restricted to the token control which is exerted over the former. Token control exists to manage the dialogue between two applications, but correct functioning of Layers 6 and 7, especially during error or recovery situations, may well require unrestricted protocol exchanges.

Thus, the optional Typed Data Transfer service is provided. It allows unconfirmed, unlimited-length, Typed SSDUs to be exchanged regardless of the availability and assignment of the data token. Typed data is subject to the same flow control as normal data, so if one is blocked, the other is also blocked.

When both typed and normal data are blocked, only expedited data can be passed.

This service is unnecessary for the RTS because RTS SS-user data *is* always application user data. Also, no unrestricted protocol exchanges are ever required between two RTSs.

d) The **Capability Data Exchange** service.
This optional service allows SS-users to exchange a limited amount of confirmed data while not within an activity. It is provided solely as a means for SS-users to exchange information about their "capability" to participate in a new activity. This service is not used by the RTS because all RTS data exchanges occur within activities.

There are three services concerned with token management:

e) The **Give Tokens** service.
This optional service allows an SS-user to surrender one or more specific tokens to the other SS-user. This service is unnecessary for the RTS because the Give Control service satisfies all RTS token transfer needs.

f) The **Please Tokens** service.
This optional service allows an SS-user to request the other SS-user to transfer one or more specific tokens to it. Since the RTS does not separate the tokens, it uses this service to request all the available tokens.

g) The **Give Control** service.
This optional service allows an SS-user to surrender all available tokens to the other SS-user. Since the RTS does not separate the tokens, this service satisfies all its token transfer needs.

There are three services concerned with synchronization and resynchronization:

h) The **Minor Synchronization Point** service.
This optional service allows the SS-users to separate the flow of one-way normal and typed SSDUs transmitted before the service was invoked from the subsequent flow of normal and typed SSDUs. To do this, it enables SS-users to define minor synchronization points in the flow of SSDUs. These minor synchronization points may optionally be confirmed, but have no implications on the data flow.

Minor synchronization points are identified by synchronization point serial numbers. The serial number is incremented by one each time a minor synchronization point is placed in the data flow, and each time a minor synchronization point is received, so that both SS-users have the same serial numbers for the same synchronization point. Use of this service is controlled by the synchronize-minor token.

This service is very important to the RTS. RTS APDUs can be very long, because an entire (even multi-page) user message is carried in a single APDU. The use of synchronization points within these APDUs minimizes the retransmission required after errors and is therefore important in keeping down transmission overheads.

i) The **Major Synchronization Point** service.
This optional service allows the SS-users to confine the flow of full-duplex, sequentially transmitted normal, typed and expedited SSDUs in each direction within a dialogue unit. To do this, it enables SS-users to define major synchronization points in the flow of SSDUs. A major synchronization point must be confirmed before the requesting SS-user is permitted to send any subsequent data, thereby clearly separating the data flow before and after the major synchronization point into dialogue units. Use of this service is controlled by the major/activity token.

This service is not required by the RTS because the RTS does not use full-duplex data transmission, typed or expedited data. Also, the RTS does not need to structure its dialogue into dialogue units because RTS dialogue context remains constant, that of Message Handling.

j) The **Resynchronize** service.
This optional service allows the SS-users to re-establish communication, in an orderly manner, within the current session connection. This typically occurs following an error, lack of response by either SS-user or the SS-provider, or disagreements between SS-users. This service sets the session connection to either a previous or a new synchronization point and reassigns the available tokens. This service may cause loss of normal, typed and expedited SSDUs.

This service is not used by the RTS. Instead, it uses the form of resynchronization provided by the activity management services.

There are two services concerned with reporting errors or unanticipated situations:

k) The **Provider-Initiated Exception Reporting** service. This optional service allows the SS-provider to notify both SS-users that a service cannot be completed due to SS-provider protocol errors or exception conditions which are not covered by other services. These exception conditions are less severe than those requiring abort of the session connection and the SS-provider anticipates that the SS-users will be able to overcome the problem. This service may cause loss of normal, typed and expedited SSDUs.

   This service is not used by the RTS. Instead, the RTS assumes that the SS-provider will abort a session connection upon detecting any unrecoverable error.

l) The **User-Initiated Exception Reporting** service. This optional service allows an SS-user to report an exception condition when the data token is available but not assigned to the SS-user. By initiating this service, the sending SS-user indicates a problem less severe than one requiring abort of the session connection. The sending SS-user anticipates that the receiving SS-user can overcome this problem and allows it to take the most appropriate course of action. This service may cause loss of normal, typed and expedited SSDUs. This service is used by the RTS.

There are five optional services concerned with activities. All these services are controlled by the major/activity token:

m) The **Activity Start** service.
This service allows an SS-user to indicate the start of a new activity.

n) The **Activity Resume** service.
This service allows an SS-user to indicate that a previously interrupted activity is re-entered.

o) The **Activity Interrupt** service.
This service allows an SS-user to abnormally terminate an activity with the implication that the work so far achieved is not to be discarded and may be resumed later. This service may cause loss of undelivered normal, typed and expedited SSDUs.

p) The **Activity Discard** service.
This service allows an SS-user to abnormally terminate an activity with the implication that the work so far achieved is to be discarded (not controlled by the SS-provider), and not resumed. This service may cause loss of undelivered normal, typed and expedited SSDUs.

q) The **Activity End** service.
This service allows an SS-user to indicate the normal end of an activity.

These activity management services are very important to X.400 because they provide reliability of data transfer.

X.400 APDUs are transferred within activities. Since the local session entity confirms delivery of activities, the delivery of X.400 APDUs are implicitly confirmed. This allows many X.400 application protocol elements to be unconfirmed, leading to simple, efficient X.400 application protocols.

Using the activity services may lead to a state where no activity is in progress on the session connection. When the RTS enters this state, it may invoke only the following services:

Activity Start,
Activity Resume,
Please Tokens,
Give Control,
Normal Data Transfer,
User Abort, and
Orderly Release.

4.7.3 The session connection release phase

This phase is concerned with releasing a previously established session connection. It has three services associated with it:

a) The **Orderly Release** service.
This service is always provided on all session connections. It allows either SS-user to release a session connection in an orderly manner. This is done cooperatively between the two SS-users without loss of data, after all in-transit data has been delivered and accepted by both SS-users.

b) The **User-Initiated Abort** service.
This service is always provided on all session connections. It allows either SS-user to initiate immediate release of a session connection and have the other SS-user informed of the release. This service terminates any outstanding service request and causes loss of all undelivered data.

c) The **Provider-Initiated Abort** service.
This service is always provided on all session connections. It allows the SS-provider to indicate to both SS-users the immediate release of a session connection for internal reasons. This service terminates any outstanding service request and causes loss of all undelivered data.

4.8 Functional units and subsets

4.8.1 Functional units

Since there are a great many optional services provided by the session layer, and as the set of services needed will vary from application to application, so many session layer implementations will only implement the subset of services needed to support the applications that have been implemented. By negotiating the set of services that are needed for the connection at establishment time, the situation is avoided whereby a connection is established and only later is it discovered that it cannot be used.

SS-user service requirements are negotiated during the session connection establishment phase in terms of functional units. Functional units are logical groupings of related services.

Certain functional units require the availability of a particular token. This implies that the functional units selected for use on a session connection determine which tokens are available on that connection. Functional units requiring a token also include those services necessary to request and transfer that token.

Table 4.2 lists all the functional units. For each it specifies its standard abbreviated name, the services associated with it, the tokens associated with it and pre-requisite functional units that must be selected with it. The last column indicates which functional units are mutually-exclusive, i.e., which functional units may not be selected together on the same session connection. Those functional units, services and tokens used by the RTS are indicated in **bold** font.

Table 4.2 Functional units

Functional unit	abbrv	Services	tk	prq	X
Kernel (non-negotiable, always provided)		**Session Connection** **Normal Data** **Orderly Release** **U-Abort** **P-Abort**			
Negotiated Release	NR	Orderly Release Give Tokens Please Tokens	tr		
Half-duplex	**HD**	Give Tokens **Please Tokens**	**dk**		X
Duplex	FD				X
Expedited Data	EX	Expedited Data			
Typed Data	TD	Typed Data			
Capability Data	CD	Capability Data		ACT	
Minor Synchronize	**SY**	**Minor Sync Point** Give Tokens **Please Tokens**	**mi**		
Major Synchronize	MA	Major Sync Point Give Tokens Please Tokens	**ma**		
Resynchronize	RESYN	Resynchronize			
Exceptions	**EXCEP**	P-Exception Report **U-Exception Report**		HD	
Activity Management	**ACT**	**Activity Start** **Activity Resume** **Activity Interrupt** **Activity Discard** **Activity End** Give Tokens **Please Tokens** **Give Control**	**ma**		

Table 4.2 highlights three important points:

a) although the P-Exception Reporting service is included in the Exceptions functional unit, the RTS does not use this service;
b) although the Give Tokens service is included in the Half-duplex, Minor Synchronize and Activity Management function units, the RTS does not use this service; and
c) the release token is never available to the RTS because the RTS does not use the Negotiated Release functional unit.

4.8.2 Subsets

A subset is a combination of the Kernel functional unit together with any other set of functional units provided that:

a) if the Capability Data functional unit is included, then the Activity Management functional unit is also included;
b) if the Exceptions functional unit is included, then the Half-duplex functional unit is also included.

Three subsets are defined:

a) the Basic Combined Subset (BCS);
b) the Basic Synchronized Subset (BSS);
c) the Basic Activity Subset (BAS).

The only subset which includes all the functional units required by the RTS is the BAS. It consists of the following functional units:

a) Kernel;
b) Half-duplex;
c) Minor Synchronize;
d) Exceptions;
e) Activity Management are used by the RTS, while:

f) Capability Data;
g) Typed Data are not used by the RTS.

Subsets are not negotiated and their only use is to provide convenient human labels for the grouping of functional units. No real use has been made of these subsets and they are therefore of no significance. They are, however, included here for completeness.

4.9 Quality of session service

The term "quality of session service" (QOSS) refers to certain characteristics of a session connection as observed between session connection endpoints. These characteristics are attributable solely to the SS-provider and are therefore independent of SS-user behaviour. Once a session connection is established, the SS-users at the two ends have the same knowledge and understanding of what the QOSS over the session connection is.

QOSS is described in terms of a well-defined set of parameters. These definitions provide both SS-users and the SS-provider with a common understanding of QOSS characteristics.

Two types of QOSS parameters are identified:

a) Those which are negotiated during the session connection establishment phase. These are:

    1) Session Connection Protection;
    2) Session Connection Priority;
    3) Residual Error Rate;
    4) Throughput, for each direction of transfer;
    5) Transit Delay, for each direction of transfer;
    6) Optimized Dialogue Transfer;
    7) Extended Control.

b) Those which are not negotiated during the session connection establishment phase but whose values are selected and/or known by some other, unspecified, methods. These are:

    1) Session Connection Establishment delay;
    2) Session Connection Establishment Failure Probability;
    3) Transfer Failure Probability;
    4) Session Connection Release Delay;
    5) Session Connection Release Failure Probability;
    6) Session Connection Resilience.

Once the session connection is established, the selected QOSS parameters are not renegotiated during its lifetime. Changes in QOSS during a session connection are not signalled to the SS-user by the SS-provider.

Clearly, most QOSS parameter values are highly dependent on the characteristics of an eventual, real session layer implementation. Therefore, possible choices and default values for each parameter will normally be specified at the time of initial SS-provider installation.

The definitions of, and negotiation procedures for the QOSS parameters are not included in this section. A thorough

treatment of these relatively complex issues will add little to the aims of this thesis. In addition, the current area of QOSS standardization is unstable as work is still in progress by the international standards bodies to provide an integrated treatment of QOS across all layers of the OSI Reference Model. This will ensure that the individual treatments in each layer satisfy overall QOS objectives in a consistent manner. For further information regarding QOSS, the reader is referred to CCITT Recommendation X.215 section 10.

There are, however, two QOSS parameters which are of interest to the RTS. They are Extended Control and Optimized Dialogue Transfer. The Extended Control parameter relates to the behaviour of the session service when normal data transfer is blocked by flow control. This parameter allows the SS-user to specify that in these circumstances it requires use of the Resynchronize, Abort, Activity Interrupt and Activity Discard services. This parameter was inserted into the QOSS parameters as a means to indicate that the Transport Expedited Data service should be provided since the session protocol uses this to send some service primitives. If the Expedited Data functional unit is selected, the Extended Control QOSS is always provided to SS-users. This feature is not required by the RTS because it does not use the Session Expedited Data service.

The other QOSS parameter, Optimized Dialogue Transfer, permits the concatenated transfer of certain session service requests. How the SS-provider achieves this concatenation is a local implementation matter. This QOSS parameter invokes the SS-provider's Extended Concatenation Protocol option. This feature is not required by the RTS.

4.10 Introduction to session service primitives

4.10.1 Summary of primitives

Each session service is achieved by invoking a sequence of session service primitives. Tables 4.3 to 4.5 list, for each phase of the session service, those services and associated primitives used by the RTS. The standard abbreviated name for each primitive is also indicated.

Table 4.3 Connection establishment phase primitives

Service	Primitives	abbrv
Session Connection	S-CONNECT.request	SCONreq
	S-CONNECT.indication	SCONind
	S-CONNECT.response (accept)	SCONrsp+
	S-CONNECT.response (reject)	SCONrsp-
	S-CONNECT.confirm (accept)	SCONcnf+
	S-CONNECT.confirm (reject)	SCONcnf-

Table 4.4 Data transfer phase primitives

Service	Primitives	abbrv
Normal Data Transfer	S-DATA.request	SDTreq
	S-DATA.indication	SDTind
Please Tokens	S-TOKEN-PLEASE.request	SPTreq
	S-TOKEN-PLEASE.indication	SPTind
Give Control	S-CONTROL-GIVE.request	SCGreq
	S-CONTROL-GIVE.indication	SCGind
Minor Sync Point	S-SYNC-MINOR.request	SSYNmreq
	S-SYNC-MINOR.indication	SSYNmind
	S-SYNC-MINOR.response	SSYNmrsp
	S-SYNC-MINOR.confirm	SSYNmcnf
U-Exception Report	S-U-EXCEPTION-REPORT.request	SUERreq
	S-U-EXCEPTION-REPORT.indication	SUERind
Activity Start	S-ACTIVITY-START.request	SACTSreq
	S-ACTIVITY-START.indication	SACTSind
Activity Resume	S-ACTIVITY-RESUME.request	SACTRreq
	S-ACTIVITY-RESUME.indication	SACTRind
Activity Interrupt	S-ACTIVITY-INTERRUPT.request	SACTIreq
	S-ACTIVITY-INTERRUPT.indication	SACTIind
	S-ACTIVITY-INTERRUPT.response	SACTIrsp
	S-ACTIVITY-INTERRUPT.confirm	SACTIcnf
Activity Discard	S-ACTIVITY-DISCARD.request	SACTDreq
	S-ACTIVITY-DISCARD.indication	SACTDind
	S-ACTIVITY-DISCARD.response	SACTDrsp
	S-ACTIVITY-DISCARD.confirm	SACTDcnf
Activity End	S-ACTIVITY-END.request	SACTEreq
	S-ACTIVITY-END.indication	SACTEind
	S-ACTIVITY-END.response	SACTErsp
	S-ACTIVITY-END.confirm	SACTEcnf

Table 4.5 Session connection release phase primitives

Service	Primitives	abbrv
Orderly Release	S-RELEASE.request	SRELreq
	S-RELEASE.indication	SRELind
	S-RELEASE.response (accept)	SRELrsp+
	S-RELEASE.response (reject)	**SRELrsp-**
	S-RELEASE.confirm (accept)	SRELcnf+
	S-RELEASE.confirm (reject)	**SRELcnf-**
User Abort	S-U-ABORT.request	SUABreq
	S-U-ABORT.indication	SUABind
Provider Abort	S-P-ABORT.indication	SPABind

Note:

Although the S-RELEASE.response (reject) and S-RELEASE.confirm (reject) primitives are part of the Orderly Release service, they may never be used by the RTS, implying that the RTS may never reject a release request. The reason for this is clarified in the next subsection.

4.10.2 Token restrictions on sending primitives

Table 4.6 specifies the token restrictions under which those service primitives used by the RTS, and requiring tokens, may be issued. The columns under 'tokens' specify the usual token restrictions. By combining these restrictions with the extra token restrictions imposed by the RTS, namely that the release token is never available and the available tokens are never separated, the last column, 'RTS', is derived. It specifies the token restrictions specific to the RTS.

Table 4.6 Token restrictions on service primitives

Service primitives	tokens				RTS
	dk	mi	ma	tr	
S-RELEASE.request	2	2	2	2	snd
S-RELEASE.response (reject)	nr	nr	nr	0	---
S-DATA.request (half-duplex)	1	nr	nr	nr	snd
S-TOKEN-PLEASE.request (dk)	0	nr	nr	nr	rcv
S-TOKEN-PLEASE.request (mi)	nr	0	nr	nr	rcv
S-TOKEN-PLEASE.request (ma)	nr	nr	0	nr	rcv
S-TOKEN-PLEASE.request (tr)	nr	nr	nr	0	---
S-CONTROL-GIVE.request	2	2	1	2	snd
S-SYNC-MINOR.request	2	1	nr	nr	snd
S-U-EXCEPTION-REPORT.request	0	nr	nr	nr	rcv
S-ACTIVITY-START.request	2	2	1	nr	snd
S-ACTIVITY-RESUME.request	2	2	1	nr	snd
S-ACTIVITY-INTERRUPT.request	nr	nr	1	nr	snd
S-ACTIVITY-DISCARD.request	nr	nr	1	nr	snd
S-ACTIVITY-END.request	2	2	1	nr	snd

Key:

Usual token restrictions:
 0: Token available and not assigned to the SS-user.
 1: Token available and assigned to the SS-user.
 2: Token not available or token assigned to the SS-user.
nr: No restriction.

Usual token restrictions plus RTS restrictions:
snd: All available tokens assigned to the RTS, i.e., only the sending RTS may issue this primitive.
rcv: All available tokens not assigned to the RTS, i.e., only the receiving RTS may issue this primitive.
---: The primitive may never be issued by either RTS.

4.10.3 Sequencing of primitives

All RTS request and response primitives are delivered by the SS-provider in the order in which they are submitted by the RTS, except for the following:

a) S-ACTIVITY-INTERRUPT;
b) S-ACTIVITY-ABORT;
c) S-U-ABORT.

These may be delivered earlier than previously submitted primitives, but not later than subsequently submitted primitives.

4.10.4 Serial number management

Certain primitives carry a synchronization point serial number, which is used to identify a synchronization point. Synchronization points are assigned valid numbers in the range 0 to 999998 by the SS-provider. It is the responsibility of the SS-user to ensure that the number assigned by the SS-provider in a synchronization point request does not exceed 999998.

The synchronization point serial number 999999 is also valid for use by the RTS, but only in the Session Connection service, which requires the synchronization point serial number of the next synchronization point.

The management of synchronization point serial numbers is defined in terms of:

a) operations on abstract local variables managed by the SS-provider, and
b) primitives issued by the SS-user in order to invoke these operations.

Of the four abstract local variables available, the RTS requires only the following three:

a) **V(M)** is next serial number to be used.

b) **V(A)** is the lowest serial number to which a synchronization point confirmation is expected. No confirmation is expected when V(A) = V(M).

c) **Vsc** indicates whether or not the SS-user has the right to issue minor synchronization point responses. Vsc has the following values:

Vsc = true:
the SS-user has the right to issue minor synchronization point responses when V(A) is less than V(M).

Vsc = false:
the SS-user does not have the right to issue minor synchronization point responses.

The following services used by the RTS affect these variables:

a) Session Connection Establishment;
b) Minor Synchronization Point;
c) Activity Start;
d) Activity Resume;
e) Activity End.

Tables 4.7 defines the operations on local variables used by the RTS, as invoked by primitives used by the RTS.

Table 4.7 Operations on variables

Events	condition for valid primitive	condition for update of variable	update of variables		
			V(A)	V(M)	Vsc
SSYNmreq SACTEreq	sn = V(M)	Vsc true	V(M)	V(M)+1	false
		Vsc false		V(M)+1	false
SACTEind	sn = V(M)	Vsc true		V(M)+1	
		Vsc false	V(M)	V(M)+1	
SSYNmind	sn = V(M)	Vsc true		V(M)+1	true
		Vsc false	V(M)	V(M)+1	true
SACTErsp	sn = V(M)-1		V(M)		
SACTEcnf	sn = V(M)-1		V(M)		
SSYNmrsp	Vsc = true & V(M)>sn>=V(A)*		sn+1		
SSYNmcnf	Vsc = false & V(M)>sn>=V(A)		sn+1		
SACTRreq SACTRind	sn >= 0 & sn <= 999998		sn+1	sn+1	
SACTSreq SACTSind			1	1	
SCONrsp+ SCONcnf+		sn present	sn	sn	false

Key:

sn: synchronization point serial number quoted in session service primitive.

*: sn not equal to V(M)-1 if activity end outstanding.

4.11 Session services and primitives used by the RTS

This section describes in detail the services and their associated primitives as used by the RTS.

Each service description starts by stating which RTS (the sender or the receiver) may request (invoke) the service and when it may do so. This is followed by a brief description of what the RTS uses the service for. Although this information has no impact on the SS-provider, it does provide the reader with additional insight into the operation and application of the service. More general aspects of the service are then described, followed by a table showing the types of primitives associated with the service and the parameters associated with each primitive type. The primitive types are identified by the following abbreviations:

abbreviation	primitive type
req	request
ind	indication
rsp	response
cnf	confirm

The presence of a parameter in a primitive is indicated by means of the following key:

M: presence of the parameter is mandatory.
C: presence of the parameter is conditional.
U: presence of the parameter is a user option.
Blank: the parameter is absent.
(=): the parameter value is identical to that in the preceding primitive.

A detailed description of each parameter and its use by the RTS is then given. Depending on whether a parameter is transparent

to the SS-provider or not, its use by the RTS may have implications for the SS-provider.

Time sequence diagrams defining the sequence of primitives for successful service invocations are not given here because they are readily implied by the description of the services and their primitive types. These diagrams may be found in CCITT Recommendation X.215 sections 12 to 14.

4.11.1 Session Connection service

This service may be invoked by either RTS at any time. The RTS invokes this service to establish a session connection with a remote RTS for the purpose of transferring X.400 APDUs.

Table 4.8

Session connection primitives and parameters

Primitive: S-CONNECT				
Parameters	req	ind	rsp	cnf
Session connection identifier	M	M(=)	M	M(=)
Calling SSAP address	M	M(=)		
Called SSAP address	M	M(=)	M(=)	M(=)
Result			M	M(=)
Quality of service	M	M	M	M
Session requirements	M	M(=)	M	M(=)
Initial sync point serial no.	C	C(=)	C	C(=)
Initial token assignments	C	C(=)	C	C(=)
SS-user data	U	C(=)	U	C(=)

Session connection identifier
is provided by the SS-users to enable them to identify the session connection. Its use is entirely at the discretion of the SS-users and it is therefore transparent to the SS-provider. It consists of the following components:

a) **Calling SS-user reference** (request and indication only) with a maximum length of 64 bytes;

b) **Called SS-user reference** (response and confirmation only) with a maximum length of 64 bytes;

c) **Common reference** with a maximum length of 64 bytes;

d) **Additional reference information** with a maximum length of 4 bytes.

The initiating RTS will supply this parameter, using it to uniquely identify the connection. It treats inclusion of the Additional Reference Information component as an option. This parameter is returned unchanged by the responding RTS, except that the Calling SS-user Reference supplied by the initiator is conveyed as the Called SS-user Reference.

Calling and Called SSAP addresses
uniquely identify the local and remote SS-users between which the session connection is to be established. The session service does not specify the content of these address fields (neither does the session protocol). The called SSAP address in the response/confirmation must be equal to the called SSAP address in the request/indication, thus not allowing for generic addressing or for call redirection.

X.400 uses a one-to-one mapping between session and transport addresses, therefore session (SSAP) addresses are equivalent to transport (TSAP) addresses.

Result
indicates the success or failure of the connection establishment request. Its value may be one of:

a) Accept.

b) Reject by the called SS-user, where the reason is one of:

   1) reason not specified;
   2) SS-user congested;
   3) the user data field provides more information.

c) Reject by the SS-provider, where the reason is one of:

   1) reason not specified;
   2) SS-provider congested;
   3) called SSAP address unknown;
   4) called SS-user not attached to SSAP.

   Reasons 3) and 4) may be regarded as persistent.

Only values a) or b) may be present in a response, while any values may be present in a confirm.

Quality of service
is a list of parameters which are defined and negotiated as described in CCITT Recommendation X.215 section 10.

The RTS sets the components Extended Control and Optimized Dialogue Transfer to "not required". These values indicate that the Transport Expedited Data Transfer service and the SS-provider Extended Concatenation Protocol option are not

required. The RTS sets the remaining parameters such that default values are used.

Session requirements
is a list of functional units required by the SS-user, subject to the following restrictions and negotiation rules:

The Kernel functional unit is always used. Each SS-user proposes the use or non-use of each of the other functional units. A functional unit is selected only if both SS-users propose it and it is supported by the SS-provider.

The requestor and acceptor may propose any functional units, provided that:

a) if the Capability Data functional unit is proposed, the Activity Management functional unit is also proposed;

b) if the Exceptions functional unit is proposed, the Half-duplex functional unit is also proposed.

In addition to these restrictions, the acceptor may not propose both the Half-duplex and Duplex functional units, although at least one of them must have been proposed by the requestor.

Both the calling and the called RTS always propose (and therefore always select) only the following functional units:

a) Half-duplex;
b) Exceptions;
c) Minor Synchronize;
d) Activity Management.

Initial synchronization point serial number
identifies the initial synchronization point. Its value is in the range 0 to 999999.

The SS-users must only propose values for this parameter if they propose any of the Major Synchronize, Minor Synchronize or Resynchronize functional units, but do not propose the Activity Management functional unit. The reason for this is that, if the Activity Management functional unit is selected, the initial synchronization point serial number will always be set to one by the Activity Start service, irrespective of the values proposed here by the SS-users.

Although the RTS always proposes and selects the Activity Management functional unit, it still proposes a value for this parameter, always proposing 0. As explained above, this value will have no effect on the actual initial synchronization point serial number used, which will always be 1.

Initial token assignments
is a list of the initial sides to which the available tokens are assigned. It is only required if the corresponding tokens are available. Exactly which tokens are available on a session connection is determined by which functional units are selected for use on the session connection, as explained earlier.

When the calling SS-user proposes a functional unit that requires a token, it also proposes the initial token setting in a request/indication, which may be one of:

a) requestor (calling SS-user) side;
b) acceptor (called SS-user) side;
c) acceptor (called SS-user) choice.

Only if the functional unit is then selected and the calling SS-user proposed c), will the acceptor reply in a response/confirm with either a) or b). Otherwise, the parameter in a response/confirm is absent.

If use of the functional unit is then selected, the token is assigned to:

a) the side proposed by the called SS-user if "acceptor choice" was proposed by the calling SS-user; or

e) in all other cases, the side proposed by the calling SS-user.

Since only the data, minor-synchronize and major/activity tokens are available to the RTS (which never separates them), the RTS always assigns them all to the same side. This allows the RTS to obtain either a one-way monologue or two-way alternate dialogue session.

SS-user data
contains up to 512 bytes of user data. It is transparent to the SS-provider.

The correspondent RTSs use this parameter to negotiate various additional, application-specific, connection parameters between themselves. Although the data are transparent to the SS-provider, one of these parameters is of indirect importance to the SS-provider. This parameter is the *checkpointSize* parameter. The RTSs use it to negotiate the maximum amount of data (in units of 1024 bytes) that may be sent between two minor synchronization points. Effectively, this is the maximum SSDU size. A value of zero indicates that no checkpointing (minor synchronization points) will be done, indicating unlimited SSDU size. This information will be important when a real session layer implementation is considered.

4.11.2 Normal Data transfer service

This service may only be invoked by the sending RTS when an activity is in progress. The sending RTS transfers an entire X.400 APDU within an activity as X.400 APDU segments in one or more normal SSDUs. These normal SSDUs are carried by this service.

Table 4.9

Normal data transfer primitives and parameters

Primitive: S-DATA		
Parameters	req	ind
SS-user data	M	M(=)

SS-user data
is a normal SSDU. It is transparent to the SS-provider. Its size is an integral number of bytes greater than zero and unlimited in length.

As stated earlier, the maximum SSDU size for RTS use will have been negotiated between the RTSs by means of the *checkpointSize* parameter. The sending RTS will submit, in S-DATA.requests, SSDUs that conform to that agreement.

If the selected value for *checkpointSize* was zero, the SSDU will contain an entire, unlimited-length, X.400 APDU. This implies that an X.400 APDU will be transferred in an activity as a single SSDU.

If the selected value for *checkpointSize* was non-zero, the SSDU will contain either an entire X.400 APDU or a segment thereof of size equal to or smaller than the selected *checkpointSize*. This implies that, if the X.400 APDU must

be segmented, it will be transferred in an activity as a series of SSDUs, the maximum size of each being the selected *checkpointSize*.

4.11.3 Please Tokens service

This service may only be invoked by the receiving RTS, at any time. Upon receipt of the indication, the sending RTS invokes the Give Control service, thereby passing all the available tokens, and therefore control of the session connection, to the receiving RTS.

Table 4.10

Please tokens primitives and parameters

Primitive: S-TOKEN-PLEASE		
Parameters	req	ind
Tokens SS-user data	M U	M(=) C(=)

Tokens
is a list of available tokens not assigned to but requested by the SS-user. The receiving RTS will only request the data token. Since the RTS never separates the tokens, the sending RTS always surrenders all the available tokens when it invokes the Give Control service.

SS-user data
contains up to 512 bytes of user data. It is transparent to the SS-provider. It is used by the RTS to carry a priority parameter.

4.11.4 Give Control service

This service may only be invoked by the sending RTS when no activity is in progress. The sending RTS requests this service to pass all available tokens (data, minor-synchronize and major/activity), and therefore control of the session connection, to the receiving RTS.

Table 4.11

Give control primitives and parameters

Primitive: S-CONTROL-GIVE		
Parameters	req	ind
none.		

4.11.5 Minor Synchronization Point service

This service may only be invoked by the sending RTS while an activity is in progress.

As stated earlier, the RTSs will only use this service if the *checkpointSize* parameter they negotiated is greater than zero. The sending RTS then inserts a minor synchronization point after each S-DATA.request (SSDU) sent within an activity. The sending RTS interprets a confirmed minor synchronization point as signifying that the SSDU has been secured by the receiving RTS and therefore does not require retransmission. If the receiving RTS has detected a problem, it does not confirm the minor synchronization point, but invokes the User Exception Reporting service or the User Abort service, depending on the severity of the problem.

The requestor may request explicit confirmation of a minor synchronization point, and the sending RTS always does. However, the SS-provider does not require that an explicit confirmation be issued. The acceptor may issue a confirmation even if explicit confirmation is not requested.

Responses are issued in the order in which the corresponding indications were received. A further minor synchronization point request may be made while previous minor synchronization points are unconfirmed.

The confirmation of a minor synchronization point confirms all previously unconfirmed minor synchronization points. The number of unconfirmed minor synchronization points is not limited by the SS-provider.

Any semantics associated with request and confirmation of a minor synchronization point are transparent to the SS-provider.

Table 4.12

Minor synchronization point primitives and parameters

Primitive: S-SYNC-MINOR				
Parameters	req	ind	rsp	cnf
Type	M	M(=)		
Sync point serial number	M	M(=)	M	M(=)
SS-user data	U	C(=)	U	C(=)

Type

indicates whether or not explicit confirmation is requested by the SS-user. It is transparent to the SS-provider and its value is one of:

a) explicit confirmation;
b) optional confirmation.

The sending RTS always requests explicit confirmation. Each minor synchronization point must be confirmed, in the order received, by the receiving RTS.

Synchronization point serial number
identifies the minor synchronization point. Its value is in the range 0 to 999998. Its use is defined in Table 4.7.

SS-user data
contains up to 512 bytes of user data. It is transparent to the SS-provider. It is not used by the RTS.

4.11.6 User Exception Reporting service

This service may only be invoked by the receiving RTS while an activity is in progress. The receiving RTS requests this service if it detects a non-severe problem which it anticipates the sending RTS can overcome, and allows the sending RTS to take the most appropriate action.

Following a request, and until the error condition is cleared:

a) SSDUs will be discarded by the SS-provider;
b) synchronization point indications will not be given to the requestor;
c) the requestor is only permitted to invoke the User Abort service.

On receipt of an indication, the sending RTS may take one of the following actions (in increasing order of severity) to clear the error:

a) Interrupt the current activity by using the Activity Interrupt service.
b) Discard the current activity by using the Activity Discard service.
c) Abort the session connection by using the User Abort service.

The sending RTS may not request any other services until the error is cleared.

Table 4.13

User exception reporting primitives and parameters

Primitive: S-U-EXCEPTION-REPORT		
Parameters	req	ind
Reason SS-user data	M U	M(=) C(=)

Reason
specifies the reason for the exception report. It is transparent to the SS-provider. Its value is one of:

a) SS-user receiving ability jeopardized;
b) local SS-user error;
c) sequence error;
d) demand data token;
e) unrecoverable procedural error;
f) non-specific error.

The receiving RTS will never specify reason d) for the following two reasons:

a) Passing the data token (together with all other available tokens) to the receiving RTS will obviously not aid in completing the X.400 APDU transfer because the original sending RTS can then no longer request any data transfer services.

b) CCITT Recommendation X.215 section 13.12.1 does not recommend that the error condition be cleared by passing the data token when the Activity Management functional unit has been selected, as is the case with the RTS.

SS-user data
contains up to 512 bytes of user data. It is transparent to the SS-provider. It is not used by the RTS.

4.11.7 Activity Start service

This service may only be invoked by the sending RTS when no activity is in progress. The sending RTS requests this service to indicate that a new activity is entered, which means that a new X.400 APDU is to be sent (the RTS sends one X.400 APDU per activity). It may then immediately start sending the X.400 APDU in a S-DATA.request since the Activity Start service is not confirmed.

This service sets the value of the next synchronization point serial number to be used to one. Only one activity may exist at a time on a session connection.

Table 4.14

Activity start primitives and parameters

Primitive: S-ACTIVITY-START		
Parameters	req	ind
Activity identifier SS-user data	M U	M(=) C(=)

Activity identifier
enables the SS-users to identify the new activity. It is transparent to the SS-provider and has a maximum length of 6 bytes. Since the RTS sends one APDU per activity, this parameter effectively becomes the APDU identifier.

SS-user data
contains up to 512 bytes of user data. It is transparent to the SS-provider. It is not used by the RTS.

4.11.8 Activity Resume service

This service may only be invoked by the sending RTS when no activity is in progress. The sending RTS requests this service to continue sending an activity (an X.400 APDU) that was previously interrupted by an activity interrupt, user abort or provider abort.

Table 4.15

Activity resume primitives and parameters

Primitive: S-ACTIVITY-RESUME		
Parameters	req	ind
Activity identifier	M	M(=)
Old activity identifier	M	M(=)
Sync point serial number	M	M(=)
Old session connection identifier	U	C(=)
SS-user data	U	C(=)

Activity identifier
allows the SS-users to give a new identifier to the activity being resumed. It is transparent to the SS-provider and has a maximum length of 6 bytes.

Old activity identifier
is the original identifier of the activity being resumed. It is transparent to the SS-provider and has a maximum length of 6 bytes.

Synchronization point serial number
is the value of the next synchronization point serial number to be used minus one. Here the sending RTS supplies the serial number of the last confirmed minor synchronization point in the interrupted activity. If there was no previously confirmed minor synchronization point, the activity cannot be resumed and must therefore be discarded. Use of this parameter is defined in Table 4.7.

Old session connection identifier
is the identifier of the session connection in which the activity being resumed was originally started. The SS-user only provides this parameter if the activity being resumed was started on a different session connection. This parameter is transparent to the SS-provider and consists of:

a) **Calling SS-user reference**, with a maximum length of 64 bytes;
b) **Called SS-user reference**, with a maximum length of 64 bytes;
c) **Common reference**, with a maximum length of 64 bytes;
d) **Additional reference information**, with a maximum length of 4 bytes.

The RTS does not use the Called SS-user Reference component and treats inclusion of the Additional Reference Information component as an option.

SS-user data
contains up to 512 bytes of user data. It is transparent to the SS-provider. It is not used by the RTS.

4.11.9 Activity Interrupt service

This service may only be invoked by the sending RTS while an activity is in progress. The sending RTS may request this service if it detects a local problem which is less severe than one requiring abort of the session connection. After receipt of the confirm, all available tokens are assigned to the requestor. This has no effect on the sending RTS since it already owns all the available tokens.

After issuing a request, the requestor is unable to invoke any services, except User Abort, until the confirm is received.

After receiving an indication, the acceptor is unable to invoke any services, except User Abort, until the response is issued.

Use of this service may cause loss of data which has not yet been delivered to the receiving SS-user. Only one interrupted activity may exist at a time on a session connection.

Table 4.16

Activity interrupt primitives and parameters

Primitive: S-ACTIVITY-INTERRUPT				
Parameters	req	ind	rsp	cnf
Reason	U	C(=)		

Reason
specifies the reason for the activity interrupt. It is transparent to the SS-provider. Its value is one of:

a) SS-user receiving ability jeopardized;
b) local SS-user error;
c) sequence error;
d) demand data token;
e) unrecoverable procedural error;
f) non-specific error.

The sending RTS will never specify reason d) because it already owns the data token.

4.11.10 Activity Discard service

This service may only be invoked by the sending RTS while an activity is in progress. The sending RTS may request this service if it detects a local problem which is less severe than one requiring abort of the session connection. After receipt of the confirm, all available tokens are assigned to the requestor. This has no effect on the sending RTS since it already owns all the available tokens.

After issuing a request, the requestor is unable to invoke any services, except User Abort, until the confirm is received.

After receiving an indication, the acceptor is unable to invoke any services, except User Abort, until the response is issued.

Use of this service may cause loss of data which has not yet been delivered to the receiving SS-user.

Table 4.17

Activity discard primitives and parameters

Primitive: S-ACTIVITY-DISCARD				
Parameters	req	ind	rsp	cnf
Reason	U	C(=)		

Reason
specifies the reason for the activity discard. It is transparent to the SS-provider. Its value is one of:

a) SS-user receiving ability jeopardized;
b) local SS-user error;
c) sequence error;
d) demand data token;
e) unrecoverable procedural error;
f) non-specific error.

The sending RTS will never specify reason d) because it already owns the data token.

4.11.11 Activity End service

This service may only be invoked by the sending RTS while an activity is in progress. The sending RTS requests this service to mark the end of the transmitted X.400 APDU. Once successfully confirmed, it indicates to both RTSs that the X.400 APDU has been successfully transferred and secured.

After issuing a request, in addition to any existing restrictions, the sending RTS is unable to invoke any services, except:

User Abort,
Activity Interrupt, or
Activity Discard

until the confirm is received.

After receiving an indication, in addition to any existing restrictions, the receiving RTS may not invoke the following services:

Minor Synchronization Point,
Activity Interrupt,
Activity Discard,
Activity End, or
Orderly Release

until the response is issued.

The sending RTS may not invoke any services, except:

Activity Start,
Activity Resume,
Please Tokens,
Give Control,
Normal Data Transfer,
Orderly Release or
User Abort

until an activity is started or resumed.

Table 4.18

Activity end primitives and parameters

Primitive: S-ACTIVITY-END				
Parameters	req	ind	rsp	cnf
Sync point serial number SS-user data	M U	M(=) C(=)	 U	 C(=)

Synchronization point serial number
is the serial number of an implied major synchronization point. Since the Major Synchronization Point service is not used by the RTS, the value of this parameter has no effect on RTS operation. Use of this parameter is defined in Table 4.7.

SS-user data
contains up to 512 bytes of user data. It is transparent to the SS-provider. It is not used by the RTS.

4.11.12 Orderly Release service

This service may only be invoked by the sending RTS when no activity is in progress. The receiving RTS cannot refuse the release because the release token is never available to the RTS.

Table 4.19

Orderly release primitives and parameters

Primitive: S-RELEASE				
Parameters	req	ind	rsp	cnf
Result SS-user data	 U	 C(=)	M U	M(=) C(=)

Result
indicates whether or not the session release is granted. Its value may be one of:

a) affirmative;
b) negative.

The receiving RTS will always accept the release, specifying the value "affirmative", because the release token is never available to it.

SS-user data
contains up to 512 bytes of user data. It is transparent to the SS-provider. It is not used by the RTS.

4.11.13 User Abort service

Either RTS may invoke this service at any time if it detects a severe problem which it anticipates cannot be overcome. The session connection is immediately released, the other RTS is informed of the release and all undelivered data is lost.

Table 4.20

User abort primitives and parameters

Primitive: S-U-ABORT		
Parameters	req	ind
SS-user data	U	C(=)

SS-user data
contains up to 9 bytes of user data. It is transparent to the SS-provider. It is used by the RTS for sending error diagnostic information.

4.11.14 Provider Abort service

The SS-provider may invoke this service at any time to release the connection for internal reasons. The session connection is immediately released, both SS-users are informed of the release and all undelivered data is lost.

Table 4.21

Provider abort primitives and parameters

Primitive: S-P-ABORT	
Parameters	ind
Reason	M

Reason
indicates the reason for the abort. Its value is one of:

a) transport disconnect;
b) protocol error;
c) undefined.

4.12 Sequences of primitives

APPENDIX A contains state tables which define the constraints on the sequences in which the session service primitives, as used by the RTS, may occur. These constraints determine the *order* in which the RTS session service primitives may occur, but do not fully specify *when* they may occur. The issue of *when* any primitive may be issued by the RTS is a local matter for the RTS and is therefore beyond the scope of this thesis.

These state tables also specify the constraints affecting the *ability* of an SS-user or the SS-provider to issue a primitive at any particular time.

The possible sequences of primitives at one RTS session connection endpoint may be derived directly from these state tables.

4.13 Collision

A collision occurs when two correspondent SS-users issue "destructive" service requests (those which may destroy user data) simultaneously. It is the task of the SS-provider to resolve these collisions in an orderly manner.

4.13.1 Collision as viewed by the SS-user

An SS-user detects a collision when, while waiting for a confirmation of a destructive primitive, it receives an indication of another destructive primitive. Neither user can continue with two such operations simultaneously, so the SS-provider determines which colliding primitive should take precedence and which should be dropped. The primitive which is dropped may be carrying user data which will therefore be lost.

Table 4.21 defines the indications that may be received by the RTS which indicate that it has lost a collision resolved by the SS-provider.

Table 4.22 Indications resulting from collision resolution

RTS is waiting for	RTS receives		
	SACTIind	SACTDind	SUABind SPABind
Clearing error state after issuing SUERreq	X	X	X
SACTIcnf			X
SACTDcnf			X

Key:

X: Indication may be received.
Blank: Indication will not be received.

4.13.2 Collision resolution by the SS-provider

The SS-provider resolves collisions according to a set of rules. These rules allow the SS-provider and both SS-users all to have a common view of the state of the connection following a collision. In the case of colliding RTS requests, the rule is:

In the case of collision between two of the following requests, the first in the list takes precedence:

a) S-U-ABORT.request;
b) S-ACTIVITY-DISCARD.request;
c) S-ACTIVITY-INTERRUPT.request;
d) S-U-EXCEPTION-REPORT.request.

A collision between RTS requests of the same type can occur only between two S-U-ABORT.requests. The reason for this is that either RTS may issue this primitive at any time, while token restrictions ensure that only one RTS at a time may issue any other request.

5. THE SESSION PROTOCOL FOR X.400

The session protocol is a set of rules and formats (semantic and syntactic) which specifies the communication procedures of two peer session entities in the provision of session services. These procedures are specified in terms of:

a) the exchange of SPDUs between two peer session entities;
b) the exchange of session service primitives between a session entity and the SS-user in the same system;
c) the exchange of transport service primitives between a session entity and the TS-provider.

The general session protocol is intended to cater for the total range of SS-user (or application) types. Therefore, it provides many options and facilities which are structured so that subsets of protocol can be specified to cater for particular application types only.

This section specifies the session protocol subset required by the X.400 application. It derives this specification from the general session protocol specification of CCITT Recommendation X.225 [4] by tailoring the latter to provide only those services required by X.400, as defined in section 4.

5.1 Definition of terms

For the purposes of this section and the rest of this thesis, the following definitions apply:

Session Protocol Machine (SPM)
An abstract machine that performs the session protocol. A session entity is comprised of one or more SPMs, each supporting one end of a session connection and using one end of a transport connection. An SPM is therefore always attached between one SCEP and one TCEP.

X.400 SPM
An SPM which supports only the session protocol subset required by X.400. An X.400 SPM therefore represents a subset of a general SPM.

When the text refers to an 'SPM', it refers to both general and X.400 SPMs. However, when the text refers to an 'X.400 SPM', it does not necessarily include general SPMs.

local matter
A decision made by a system concerning its behaviour in the session layer that is not subject to the requirements of the session protocol.

initiator
An SPM that initiates a CONNECT SPDU.

responder
An SPM with whom an initiator wishes to establish a session connection.

sending SPM
An SPM that sends a given SPDU.

receiving SPM
An SPM that receives a given SPDU.

proposed parameter
The value for a parameter proposed by an SPM, in a CONNECT SPDU or an ACCEPT SPDU, that it wishes to use on the session connection.

selected parameter
The value for a parameter that has been chosen for use on the session connection.

valid SPDU
An SPDU which complies with the requirements of the session protocol with respect to *structure and encoding*.

invalid SPDU
An SPDU which does not comply with the requirements of the session protocol with respect to *structure and encoding*.

protocol error
Use of an SPDU that does comply with the *procedures* agreed for the session connection.

transparent data
SS-user data which is transferred intact between SPMs and which is not available for use by the SPMs.

SPDU identifier (SI)
SPDU heading information that identifies the SPDU.

parameter field
A group of one or more bytes within an SPDU used to represent a particular set of information.

length indicator (LI)
An indicator within an SPDU that specifies the length of an associated parameter field.

parameter identifier (PI)
An identifier within an SPDU which identifies the parameter in its associated parameter field.

PI unit (PIU)
An SPDU element that contains a PI field together with its associated LI field and parameter field.

parameter group identifier (PGI)
An identifier within an SPDU which identifies the parameter group in its associated parameter field. The associated parameter field may consist of a set of PI units.

PGI unit (PGIU)
An SPDU element that contains a PGI field together with its associated LI field and parameter field.

local variable
A local variable within the SPM which is used to clarify the effects of certain actions and to clarify the conditions under which certain actions are permitted.

5.2 Model of a session connection

Figure 5.1 depicts a model of a session connection. It shows, structurally, how an SPM within a session entity supports one end of a session connection. The SPM communicates with the SS-user by exchanging session service primitives through an SSAP. Similarly, the SPM communicates with the TS-provider by exchanging transport service primitives through a TSAP. The session protocol operates between peer SPMs, which exchange SPDUs by means of a transport connection and transport layer services.

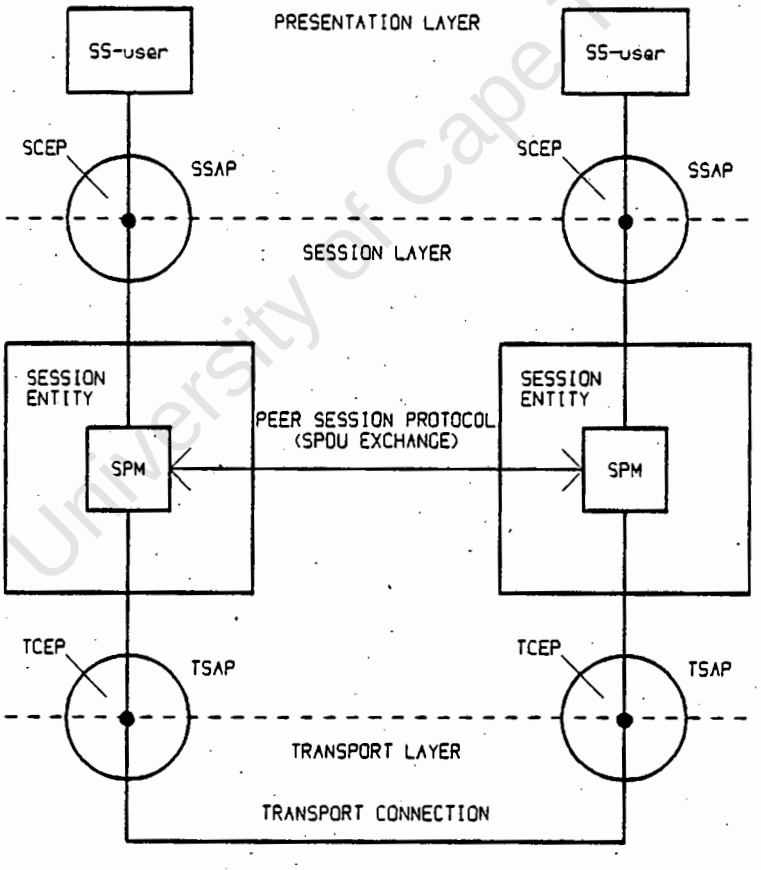

Figure 5.1 Model of a session connection

5.3 Overview of SPDUs

The exchange of session service primitives between an SPM and an SS-user will cause, or be the result of, SPDU exchanges between the SPM and its peer. Tables 5.1, 5.2 and 5.3 specify, for each of the three phases of the session service, the SPDUs associated with each session service primitive used by the RTS. These tables do not provide a complete list of all the SPDUs used by the X.400 SPM. The reason for this will be become clear in subsection 5.4, where a complete list is provided.

Table 5.1 Connection establishment phase SPDUs

Service	Primitives	Associated SPDUs
Session Connection	SCONreq	CONNECT
	SCONind	CONNECT
	SCONrsp+	ACCEPT
	SCONrsp-	REFUSE
	SCONcnf+	ACCEPT
	SCONcnf-	REFUSE

Table 5.2 Data transfer phase SPDUs

Services	Primitives	Associated SPDUs
Normal Data	SDTreq	DATA TRANSFER
	SDTind	DATA TRANSFER
Please Tokens	SPTreq	PLEASE TOKENS
	SPTind	PLEASE TOKENS
Give Control	SCGreq	GIVE TOKENS CONFIRM
	SCGind	GIVE TOKENS CONFIRM
Minor Sync Point	SSYNmreq	MINOR SYNC POINT
	SSYNmind	MINOR SYNC POINT
	SSYNmrsp	MINOR SYNC ACK
	SSYNmcnf	MINOR SYNC ACK
U-Exception Report	SUERreq	EXCEPTION DATA
	SUERind	EXCEPTION DATA
Activity Start	SACTSreq	ACTIVITY START
	SACTSind	ACTIVITY START
Activity Resume	SACTRreq	ACTIVITY RESUME
	SACTRind	ACTIVITY RESUME
Activity Interrupt	SACTIreq	ACTIVITY INTERRUPT
	SACTIind	ACTIVITY INTERRUPT
	SACTIrsp	ACTIVITY INTERRUPT ACK
	SACTIcnf	ACTIVITY INTERRUPT ACK
Activity Discard	SACTDreq	ACTIVITY DISCARD
	SACTDind	ACTIVITY DISCARD
	SACTDrsp	ACTIVITY DISCARD ACK
	SACTDcnf	ACTIVITY DISCARD ACK
Activity End	SACTEreq	ACTIVITY END
	SACTEind	ACTIVITY END
	SACTErsp	ACTIVITY END ACK
	SACTEcnf	ACTIVITY END ACK

Table 5.3 Session connection release phase SPDUs

Service	Primitives	Associated SPDUs
Orderly Release	SRELreq	FINISH
	SRELind	FINISH
	SRELrsp+	DISCONNECT
	SRELrsp-	**NOT FINISHED**
	SRELcnf+	DISCONNECT
	SRELcnf-	**NOT FINISHED**
User Abort	SUABreq	ABORT
	SUABind	ABORT
Provider Abort	SPABind	ABORT

Those service primitives and SPDUs in Table 5.3 indicated in **bold** font are not used by the X.400 SPM. Although they form part of a service used by the RTS, the RTS may never use them because the release token is never available to it.

5.4 Functional units

Table 5.4 lists all the SPDUs associated with those functional units selected by the RTS. Not all these SPDUs are used by the X.400 SPM, as will be shown. Apart from these unused SPDUs, this Table lists all the SPDUs used by the X.400 SPM.

Table 5.4 Functional units and associated SPDUs

Functional unit	SPDU name	SPDU code
Kernel (non-negotiable)	CONNECT	CN
	ACCEPT	AC
	REFUSE	RF
	FINISH	FN
	DISCONNECT	DN
	ABORT	AB
	ABORT ACCEPT (Note 1)	AA
	DATA TRANSFER	DT
Half-duplex	PLEASE TOKENS	PT
	GIVE TOKENS (Note 2)	GT
Minor Synchronize	MINOR SYNC POINT	MIP
	MINOR SYNC ACK	MIA
	PLEASE TOKENS	PT
	GIVE TOKENS (Note 2)	GT
Exceptions	EXCEPTION DATA	ED
	EXCEPTION REPORT	**ER**
Activity Management	ACTIVITY START	AS
	ACTIVITY RESUME	AR
	ACTIVITY INTERRUPT	AI
	ACTIVITY INTERRUPT ACK	AIA
	ACTIVITY DISCARD	AD
	ACTIVITY DISCARD ACK	ADA
	ACTIVITY END	AE
	ACTIVITY END ACK	AEA
	PREPARE	**PR**
	PLEASE TOKENS	PT
	GIVE TOKENS (Note 2)	GT
	GIVE TOKENS CONFIRM	GTC
	GIVE TOKENS ACK (Note 1)	GTA

Note 1:
This SPDU is not associated with a particular service primitive. Its semantics are relevant only to the SPM.

Note 2:
Although the GIVE TOKENS SPDU is usually directly associated with the Give Tokens service (which is not provided by the X.400 SPM), it is used by the X.400 SPM for Basic Concatenation purposes (see subsection 5.8.3.2).

Those SPDUs in Table 5.4 indicated in **bold** font are not used by the X.400 SPM:

a) The EXCEPTION REPORT SPDU is not used because it is associated with the Provider Exception Reporting service, which is not provided by the X.400 SPM.

b) the PREPARE SPDU is not used because it may only be sent on the Transport Expedited Data Transfer service, which is not used by the X.400 SPM (see subsection 5.8.1).

5.5 Tokens

Table 5.5 defines the token restrictions under which the X.400 SPM may send those SPDUs (and accept the associated service primitives) it uses. The columns under 'tokens' specify the usual token restrictions. By combining these restrictions with the extra restrictions imposed by X.400, namely that the release token is never available and that the tokens are never separated, the last column, 'X.400 SPM', is derived. It specifies the token restrictions specific to X.400 SPM use.

Table 5.5 Token restrictions on sending SPDUs

SPDUs	tokens			X.400 SPM
	dk	mi	ma	
FINISH	2	2	2	snd
DATA TRANSFER (half-duplex)	1	nr	nr	snd
PLEASE TOKENS (dk)	0	nr	nr	rcv
PLEASE TOKENS (mi)	nr	0	nr	rcv
PLEASE TOKENS (ma)	nr	nr	0	rcv
GIVE TOKENS CONFIRM	2	2	1	snd
MINOR SYNC POINT	2	1	nr	snd
EXCEPTION DATA	0	nr	nr	rcv
ACTIVITY START	2	2	1	snd
ACTIVITY RESUME	2	2	1	snd
ACTIVITY INTERRUPT	nr	nr	1	snd
ACTIVITY DISCARD	nr	nr	1	snd
ACTIVITY END	2	2	1	snd

Key:

Usual token restrictions:

- 0: Token available and not assigned to the SS-user who initiated the associated service primitive.
- 1: Token available and assigned to the SS-user who initiated the associated service primitive.
- 2: Token not available or token assigned to the SS-user who initiated the associated service primitive.
- nr: No restriction.

Usual token restrictions plus X.400 restrictions:

- snd: All available tokens assigned to the RTS who initiated the associated service primitive.
- rcv: All available tokens not assigned to RTS who initiated the associated service primitive.

5.6 Negotiation

During the session connection establishment phase, the correspondent SPMs negotiate the values of certain parameters which are transparent to the SS-users. These parameters and their negotiation rules are the following:

5.6.1 Negotiation of version number

Each SPM indicates all versions of the protocol that it is capable of supporting. The highest common version number is used.

5.6.2 Negotiation of maximum TSDU size

Each SPM proposes, for each direction of transfer, a maximum TSDU size that is permitted in the transport data transfer and transport connection release phases. For each pair of values for a direction of transfer, the lesser value is used.

If either SPM proposes zero, it is interpreted as unlimited TSDU length for that direction of transfer. This means that SSDUs may not be segmented into TSDUs for that direction of transfer.

5.7 Local variables

A local variable within an SPM is used to clarify the effects of certain actions and the conditions under which certain actions are permitted. The following local variables are used by the X.400 SPM:

Vact
indicates whether or not an activity is in progress, when the Activity Management functional unit has been selected. It has the following values:

Vact = true: an activity is in progress;
Vact = false: no activity is in progress.

V(M)
is the next serial number to be used.

V(A)
is the lowest serial number to which a synchronization point confirmation is expected. No confirmation is expected when V(A) = V(M).

Vsc
indicates whether or not the SS-user has the right to send minor synchronization point responses. It has the following values:

Vsc = true: the SS-user has the right when V(A) < V(M);
Vsc = false: the SS-user does not have the right.

5.8 Use of the transport service

This section specifies how the transport service, as defined in CCITT Recommendation X.214 [10], is used by the X.400 SPM. Table 5.6 lists the transport service primitives used by the X.400 SPM, together with the standard abbreviated name for each primitive.

Table 5.6. Transport service primitives

Service	Primitives	abbrv
Transport Connection Establishment	T-CONNECT.request	TCONreq
	T-CONNECT.indication	TCONind
	T-CONNECT.response	TCONrsp
	T-CONNECT.confirm	TCONcnf
Normal Data Transfer	T-DATA.request	TDTreq
	T-DATA.indication	TDTind
Transport Connection Release	T-DISCONNECT.request	TDISreq
	T-DISCONNECT.indication	TDISind

5.8.1 Transport connection establishment

This service enables two TS-users (session entities) to establish a transport connection between themselves. Simultaneous connection requests typically result in a corresponding number of transport connections. No architectural restriction is placed on the number of simultaneous transport connections associated with a TS-user or between two TS-users.

The primitives and associated parameters employed by this service are defined in Table 5.7. This is followed by a description of each parameter and its use by the X.400 SPM.

Table 5.7

Transport connection primitives and parameters

Primitive: T-CONNECT				
Parameters	req	ind	rsp	cnf
Called TSAP address	M	M(=)		
Calling TSAP address	M	M(=)		
Responding TSAP address			M	M(=)
Expedited data option	M	M(=)	M	M(=)
Quality of service	M	M	M	M(=)
TS-user data	U	C(=)	U	C(=)

Key:

M: presence of the parameter is mandatory.
C: presence of the parameter is conditional.
U: presence of the parameter is a user option.
Blank: the parameter is absent.
(=): the parameter value is identical to that in the preceding primitive.

Called TSAP address
uniquely identifies the remote TS-user to which the transport connection is to be established.

Calling TSAP address
uniquely identifies the local TS-user from which the transport connection has been requested.

Responding TSAP address
identifies the TS-user to which the transport connection has been established. Its value is identical to the Called TSAP Address parameter. This parameter may be used in future definitions of the transport service to return an address which is different from the Called TSAP Address, e.g., a specific address returned as the result of calling a generic address.

The transport service does not specify the content of these address fields.

Expedited data option
indicates whether or not the Transport Expedited Data Transfer service is to be available for the duration of the transport connection. It value is one of:

a) Expedited Data Service selected; or
b) Expedited Data Service not selected.

This service is only made available when specifically requested and agreed to by both TS-users.

The SPM will only request this service if:

a) the SS-user requested the Expedited Data functional unit; or
b) the SS-user requested an Extended Control QOSS for the session connection.

Since the RTS never requests either of these options, the X.400 SPM will never request that this service be available.

Quality of Service
is a list of parameters used by the TS-users and the TS-provider to negotiate the Quality of Transport Service (QOTS) to be available on the transport connection. These parameters are:

a) Transport Connection Establishment Delay;
b) Transport Connection Establishment Failure Probability;
c) maximum and average Throughput for each direction of transfer;
d) maximum and average Transit Delay for each direction of transfer;
e) Residual Error Rate;
f) Transfer Failure Probability;
g) Transport Connection Release Delay;
h) Transport Connection Release Failure Probability;
i) Transport Connection Protection;
j) Transport Connection Priority;
k) Transport Connection Resilience.

These parameters are defined in CCITT Recommendation X.214 section 10 and are negotiated according to the rules specified in CCITT Recommendation X.214 section 12.2.6. These issues will not be discussed here as such a discussion will be long and complex, adding little to the aims of this thesis. Suffice to say that, since the RTS specifies that default values be used for the QOSS parameters, the X.400 SPM uses default values for the QOTS parameters.

TS-user data
contains up to 32 bytes of user data. It is transparent to the TS-provider. It is not used by the X.400 SPM.

5.8.2 Reuse of the transport connection

When a session connection is refused, or has been successfully connected and subsequently disconnected by abort or orderly release, the supporting transport connection may be either disconnected or reused.

The primitives and associated parameters employed by this service are defined in Table 5.8. This is followed by a description of each parameter and its use by the X.400 SPM.

Table 5.8 Normal data transfer primitives and parameters

Primitive: T-DATA		
Parameters	req	ind
TS-user data	M	M(=)

Key:

M: presence of the parameter is mandatory.
(=): the parameter value is identical to that in the preceding primitive.

TS-user Data
is a TSDU. The maximum size of a TSDU (in bytes), for each direction of transfer, is negotiated between the correspondent SPMs during the session connection establishment phase. It may have any value greater than or equal to zero, where a zero value indicates unlimited length.

The SPM maps SPDUs into TSDUs either one-to-one or by concatenation. The maximum size of a single SPDU, or of a concatenated sequence of SPDUs, may not exceed the maximum TSDU size for that direction of transfer. There is, however, no requirement that the resulting TSDU should be of the maximum size for that direction of transfer.

5.8.3.1 Segmenting

The sending SPM maps each normal SSDU (which may be of unlimited length) one-to-one onto a DATA TRANSFER SPDU (which may carry unlimited length SSDUs), unless segmenting has been selected for that direction of transfer.

Segmenting is implicitly selected if a non-zero (i.e., limited) maximum TSDU size is negotiated for that direction of transfer. Since a maximum TSDU size necessarily constrains the maximum DATA TRANSFER SPDU size, SSDUs which are too large to fit within the constrained DATA TRANSFER SPDUs must be segmented. Each segment is then mapped onto a separate DATA TRANSFER SPDU which is mapped onto a separate TSDU. Control information in each DATA TRANSFER SPDU indicates whether it contains the first, middle or last SSDU segment.

When segmenting is being used, the receiving SPM does not deliver received data to the receiving SS-user until it has reassembled the entire SSDU from the SSDU segments it receives in consecutive DT SPDUs. Receipt of a "destructive" SPDU (EXCEPTION DATA, ACTIVITY INTERRUPT, ACTIVITY DISCARD or ABORT) before the final SSDU segment has been received causes the data received so far (and not delivered to the SS-user) to be discarded. Segmenting is transparent to the SS-users.

The reason why the session segmentation facility is provided is not clear. There is a need for an end open system to be able to control the size of data unit that it is required to handle. What is not clear is why an end open system should be capable of handling large SSDUs but not capable of handling equally large TSDUs, since the resources needed are

the same in each case. In fact, the international standards bodies have recently proposed to have this facility removed from the session protocol.

5.8.3.2 Concatenation

The sending SPM concatenates SPDUs onto TSDUs using two schemes: *basic concatenation* (mandatory) and *extended concatenation* (optional).

During session connection establishment, the SPM indicates to its peer whether or not it can accept extended concatenated SPDUs. This indication is given using the Protocol Options parameter of the CONNECT and ACCEPT SPDUs. The value of this parameter is derived from the Optimized Dialogue Transfer QOSS parameter provided by the SS-users in the S-CONNECT primitives. As explained in section 4, the RTS always sets the QOSS Optimized Dialogue Transfer parameter so that the extended concatenation feature is never invoked. Therefore, only basic concatenation must be supported by the X.400 SPM, and only this scheme will be described further.

For the purposes of basic concatenation, each SPDU belongs to one of the following three categories:

a) *Category 0 SPDUs*
   which may be mapped one-to-one onto a TSDU or may be concatenated with one Category 2 SPDU;
b) *Category 1 SPDUs*
   which are always mapped one-to-one onto a TSDU;
c) *Category 2 SPDUs*
   which are never mapped one-to-one onto a TSDU.

The categories of the SPDUs used by the X.400 SPM are listed in Table 5.9.

Table 5.9 Category 0, 1 and 2 SPDUs

Category 0 SPDUs	Category 1 SPDUs	Category 2 SPDUs
GIVE TOKENS PLEASE TOKENS	CONNECT ACCEPT REFUSE FINISH DISCONNECT ABORT ABORT ACCEPT GIVE TOKENS CONFIRM GIVE TOKENS ACK	DATA TRANSFER MINOR SYNC POINT MINOR SYNC ACK ACTIVITY START ACTIVITY RESUME ACTIVITY DISCARD ACTIVITY DISCARD ACK ACTIVITY INTERRUPT ACTIVITY INTERRUPT ACK ACTIVITY END ACTIVITY END ACK EXCEPTION DATA

Basic concatenations of a category 0 SPDU with a single category 2 SPDU, defined as valid and in the order indicated in Table 5.10, may always be mapped onto a single TSDU.

Table 5.10 Valid basic concatenation of SPDUs

First SPDU (Category 0)	Second SPDU (Category 2)
GIVE TOKENS	DATA TRANSFER
GIVE TOKENS PLEASE TOKENS	MINOR SYNC POINT MINOR SYNC ACK
GIVE TOKENS GIVE TOKENS	ACTIVITY START ACTIVITY RESUME
GIVE TOKENS PLEASE TOKENS	ACTIVITY DISCARD ACTIVITY DISCARD ACK
GIVE TOKENS PLEASE TOKENS	ACTIVITY INTERRUPT ACTIVITY INTERRUPT ACK
GIVE TOKENS PLEASE TOKENS	ACTIVITY END ACTIVITY END ACK
PLEASE TOKENS	EXCEPTION DATA

Note:

The X.400 SPM uses the GIVE TOKENS SPDU only for introducing a pair of basic concatenated SPDUs, never for passing tokens. Therefore, this SPDU will never carry any parameters.

The PLEASE TOKENS SPDU will also never carry parameters when used to introduce a pair of basic concatenated SPDUs. The reason for this is that the X.400 SPM does not allow the local concatenation of RTS service requests.

The valid mappings of SPDUs into TSDUs are illustrated in Figure 5.2.

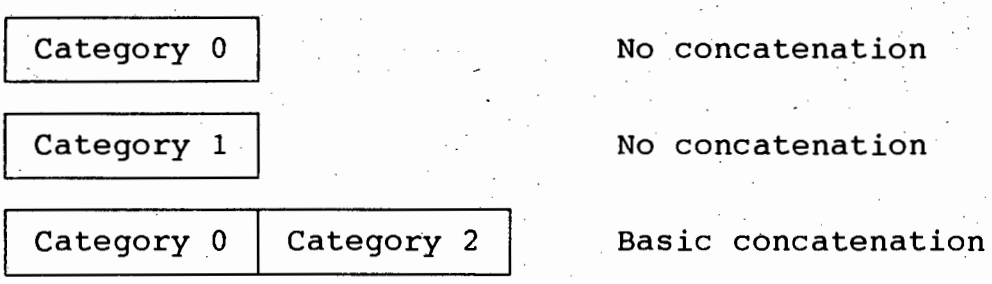

Figure 5.2 Illustration of TSDU structures

The receiving SPM is responsible for separating concatenated SPDUs and then processing them individually. Concatenation is thus transparent to the SS-users. On receipt of SPDUs that have been concatenated using basic concatenation, the category 2 SPDUs are always processed before the category 0 SPDU.

5.8.4 Transport connection release

This service is used to unconditionally release a transport connection at any time. A release request cannot be rejected. The TS-provider does not guarantee delivery of any TS-user data once this service is invoked. The release may be performed by:

a) either or both TS-users to release an established transport connection;
b) the TS-provider to release an established transport connection; all failures to maintain a transport connection are indicated in this way;
c) either or both TS-users to abandon transport connection establishment;
d) the TS-provider to indicate its inability to establish a requested transport connection.

After the session connection has been released or aborted and the transport connection is not to be reused, the transport connection is disconnected.

The primitives and associated parameters employed by this service are defined in Table 5.11. This is followed by a description of each parameter and its use by the X.400 SPM.

Table 5.11

Transport connection release primitives and parameters

Primitive: T-DISCONNECT		
Parameters	req	ind
Reason		M
TS-user data	U	C(=)

Key:

M: presence of the parameter is mandatory.
C: presence of the parameter is conditional.
U: presence of the parameter is a user option.
Blank: the parameter is absent.
(=): the parameter value is identical to that in the preceding primitive.

Reason

indicates the cause of the transport connection release. Its value may be one of:

a) Remote TS-user invoked; additional information may be given in the TS-user data parameter.
b) TS-provider invoked. This reason may be of transient or permanent nature. The following examples are given:

1) lack of local or remote resources,
2) QOTS below minimum level,
3) TS-provider error,
4) called TS-user unknown,
5) called TS-user unavailable,
6) unknown reason.

TS-user data

contains up to 64 bytes of transparent user data. It may only be present if the transport connection release was originated by a TS-user. When issuing a T-DISCONNECT.request, the SPM may optionally use this parameter to indicate the reason for the transport connection release to the remote SPM. This reason code consists of one byte with the following values:

a) 0: session protocol error for which an ABORT SPDU could not be sent;
b) 1: normal transport connection release when the transport connection is not to be reused;
c) 2: normal transport connection release when the transport connection was to be reused, but reuse was not possible for local reasons.

This parameter is not used by the X.400 SPM.

5.9 The SPDUs for X.400

This section defines the purpose, structure and encoding of each SPDU used by the X.400 SPM. For each SPDU, it provides:

a) a brief description of its purpose and use;
b) a table defining its structure and parameter fields;
c) a description of each parameter, its encoding and its use by the X.400 SPM.

This section uses the conventions and terminology established in CCITT Recommendation X.225 sections 8.1 and 8.2 for the structure and encoding of TSDUs and SPDUs. This information will therefore not be repeated here and a thorough familiarity with them will be assumed. There is, however, one convention which is unique to the Tables presented in this section:

> The inclusion of a parameter indicated in **bold** font is mandatory, while the inclusion of a parameter indicated in normal font is optional.

This section does not describe the operation of the protocol in great detail. The reason for this is that, to a large extent, there is an exact equivalence between the SPDU exchanges, parameters and restrictions and the corresponding service primitive exchanges. The valid sequences of operation of the protocol are defined in APPENDIX B in terms of state tables. These incorporate all the checks to determine the validity of a particular event at a particular time and define all protocol actions associated with protocol events.

5.9.1 CONNECT SPDU

This SPDU is sent by the initiator of the previously assigned transport connection in order to initiate a session connection.

Table 5.12 Parameters of the CONNECT SPDU

SI: 13				
Parameter Group	PGI	Parameter	PI	Length (bytes)
Connection identifier	1	Calling SS-user reference	10	64 max
		Common reference	11	64 max
		Additional reference information	12	4 max
Connect/Accept item	5	**Protocol options**	19	1
		TSDU maximum size	21	4
		Version number	22	1
		Initial serial number	23	6 max
		Token setting Item	26	1
		SS-user requirements	20	2
		Calling SSAP selector	51	16 max
		Called SSAP selector	52	16 max
User data	193			512 max

Calling SS-user reference
Common reference
Additional reference information
are all as defined by the calling SS-user.

Protocol options
indicates whether or not the initiator is able to receive extended concatenated SPDUs. The encoding for this field is:

a) bit 1 = 0 :
   able to receive extended concatenated SPDUs;
b) bit 1 = 1 :
   not able to receive extended concatenated SPDUs.

Bits 2 to 8 are reserved. The default value for this field is 1.

The X.400 SPM always sets this field to 1 because it does not support the extended concatenation protocol option.

TSDU maximum size
is present if the use of segmenting is proposed by the initiator. If present, it is encoded as follows:

a) the first two bytes contain the proposed maximum TSDU size in the direction from the initiator to the responder, encoded as a binary number, where the first byte is the high-order byte;
b) the second two bytes contain the proposed maximum TSDU size in the direction from the responder to the initiator, encoded as a binary number, where the first byte is the high-order byte.

The default value for this field is 0, indicating unlimited TSDU size in both directions of transfer and, consequently, no segmenting of normal SSDUs over the transport connection. If either pair of bytes has the value 0, there shall be no segmenting in the direction associated with the pair of bytes.

Version number
indicates the session protocol versions supported by the initiator. Bit 1 has the value 1, indicating that this version of the protocol is implemented. All other bits are reserved. The default value for this field is 1.

Initial serial number
must be present if the Activity Management functional unit is not proposed and any of the Minor Synchronize, Major Synchronize or Resynchronize functional units are proposed. As an SS-user option, it may be present if the Activity Management functional unit is proposed provided that any of the Minor Synchronize, Major Synchronize or Resynchronize functional units are also proposed.

Each digit of the serial number is encoded as a single byte, as follows:

a) 0 : 0011 0000;
b) 1 : 0011 0001;
c) 2 : 0011 0010;
d) 3 : 0011 0011;
e) 4 : 0011 0100;
f) 5 : 0011 0101;
g) 6 : 0011 0110;
h) 7 : 0011 0111;
i) 8 : 0011 1000;
j) 9 : 0011 1001.

The serial number may range from 0 to 999999. The most significant digit is encoded first in the parameter field. Leading zeros may be omitted. The default value for this field is 0.

Token setting item
indicates the initial positions of the tokens as proposed by the calling SS-user. Each token is represented by a bit pair in this field, as follows:

a) bits 8, 7 : release token;
b) bits 6, 5 : major/activity token;
c) bits 4, 3 : synchronize-minor token;
d) bits 2, 1 : data token.

The encoding for each bit pair is:

a) 00 : initiator's side;
b) 01 : responder's side;
c) 10 : called SS-user's choice;
d) 11 : reserved.

These values are only relevant if the appropriate functional units are requested in the SS-user Requirements parameter. This parameter field may be absent if no functional unit requiring a token has been requested. The default value for this field is 0, i.e., all tokens whose availability is proposed in the SS-user Requirements parameter are assigned to the Calling SS-user.

SS-user requirements
indicates the functional units proposed by the calling SS-user for use on the session connection. Each field bit represents a functional unit, as follows:

a) bit 1 : Half-duplex; (proposed by the RTS)
b) bit 2 : Full-duplex;
c) bit 3 : Expedited Data;
d) bit 4 : Minor Synchronize; (proposed by the RTS)
e) bit 5 : Major Synchronize;
f) bit 6 : Resynchronize;
g) bit 7 : Activity Management; (proposed by the RTS)
h) bit 8 : Negotiated Release;
i) bit 9 : Capability Data;
j) bit 10 : Exceptions; (proposed by the RTS)
k) bit 11 : Typed Data.

Bits 12 to 16 are reserved. The encoding for each bit is:

a) 0 : use of the functional unit is not proposed;
b) 1 : use of the functional unit is proposed.

As used by the X.400 SPM, this field will always have the binary value 0000 0010 0100 1001.

The default binary value for this field is 0000 0011 0100 1001.

Calling SSAP selector
Called SSAP selector
are derived by the SPM from the Calling and Called SSAP address parameters supplied by the calling SS-user. They are used by the session entities for hierarchical session layer addressing, identifying an SSAP within the scope of a supporting TSAP. However, since there is always a one-to-one mapping between SSAP and TSAP addresses for X.400, these selectors are not required, and may be omitted from this SPDU.

User data
contains transparent data supplied by the calling SS-user.

5.9.2 ACCEPT SPDU

This SPDU is sent by the responder if the called SS-user accepts an incoming session connection establishment attempt.

Table 5.13 Parameters of the ACCEPT SPDU

SI: 14				
Parameter Group	PGI	Parameter	PI	Length (bytes)
Connection identifier	1	Called SS-user reference	9	64 max
		Common reference	11	64 max
		Additional reference information	12	4 max
Connect/Accept item	5	**Protocol options**	19	1
		TSDU maximum size	21	4
		Version number	22	1
		Initial serial number	23	6 max
		Token setting Item	26	1
		Token item	16	1
		SS-user requirements	20	2
		Calling SSAP selector	51	16 max
		Called SSAP selector	52	16 max
User data	193			512 max

Called SS-user reference
Common reference
Additional reference information
are all as defined by the called SS-user.

Protocol options
indicates whether or not the responder is able to receive extended concatenated SPDUs. It has the same encoding, default and value as in the CONNECT SPDU.

TSDU maximum size
is present if the use of segmenting is proposed by the responder. It has the same encoding and default as in the CONNECT SPDU.

Version number
indicates the session protocol versions supported by the responder. It has the same encoding, default and value as in the CONNECT SPDU.

Initial serial number
must be present if the Activity Management functional unit is not selected and any of the Minor Synchronize, Major Synchronize or Resynchronize functional units are selected. It has the same encoding and default as in the CONNECT SPDU.

Token setting item
indicates the initial assignment of each token available on this session connection. It has the same encoding as in the CONNECT SPDU. In the case where the initial assignment of a token was indicated as called SS-user's choice (in the same field of the associated CONNECT SPDU), this field will contain the value chosen by the called SS-user. Otherwise, the values set in the CONNECT SPDU are returned. The value "called SS-user's choice" is not permitted in this SPDU. These values are only relevant if the appropriate functional units are requested in the SS-user Requirements parameter. This parameter field may be absent if no functional unit requiring a token has been requested.

Token item
indicates which tokens are requested by the called SS-user. It is encoded as follows:

a) bit 7 = 1 : release token;
b) bit 5 = 1 : major/activity token;
c) bit 3 = 1 : synchronize-minor token;
d) bit 1 = 1 : data token.

Bits 2, 4, 6 and 8 are reserved. Bits corresponding to tokens which are not available are ignored. The default value for this field is 0, i.e., no tokens are requested.

SS-user requirements
indicates the functional units proposed by the called SS-user for use on the session connection. It has the same encoding, default and value as in the CONNECT SPDU. This field may not have both bit 1 (Half-duplex functional unit) and bit 2 (Duplex functional unit) set, but the chosen bit must have been set in the CONNECT SPDU.

Calling SSAP selector
Called SSAP selector
are not used, for the same reasons as in the CONNECT SPDU.

User data
contains transparent data supplied by the called SS-user.

5.9.3 REFUSE SPDU

This SPDU is sent by the responder to reject an attempt to establish a session connection. This rejection may be by either the called SS-user or the responder itself.

Table 5.14 Parameters of the REFUSE SPDU

SI: 12				
Parameter Group	PGI	Parameter	PI	Length (bytes)
Connection identifier	1	Called SS-user reference	9	64 max
		Common reference	11	64 max
		Additional reference information	12	4 max
		Transport disconnect	17	1
		SS-user requirements	20	2
		Version number	22	1
		Reason code	50	513 max

Called SS-user reference
Common reference
Additional reference information
are all as defined by the called SS-user.

Transport disconnect
indicates whether or not the transport connection is to be kept. The encoding for this field is:

a) bit 1 = 0 : transport connection is to be kept;
b) bit 1 = 1 : transport connection is to be released.

Bits 2 to 8 are reserved. The default value for this field is 1.

SS-user requirements
may only be present if the Reason Code is 2. It indicates the functional units required by the called SS-user. It has the same encoding and default as in the CONNECT SPDU.

Version number
indicates the session protocol versions supported by the responder. It has the same encoding, default and value as in the CONNECT SPDU.

Reason code
indicates the reason for the connection rejection. It contains a reason code in the first byte. Depending on the value of this byte, up to 512 additional bytes may be included. The following values are defined for the first byte:

a) 0 : reason not specified;
b) 1 : rejection by called SS-user due to temporary congestion;
c) 2 : rejection by called SS-user; the following 512 bytes of user data provide more information;
d) 129 : SSAP selector unknown;
e) 130 : SS-user not attached to SSAP;
f) 131 : SPM congestion at connect time;
g) 132 : proposed protocol versions not supported.

All other values are reserved. The default value for this field is 0.

Reasons d), e) and g) may be reported to the calling SS-user as persistent, others reported as transient.

Reason d) is never used by the X.400 SPM because it never uses SSAP selectors.

Reason e) implies that the called SS-user did not respond to an SCONind primitive issued to it by the responder, probably within some (undefined) timeout period.

Reason f) indicates that the responder cannot support any more session connections at the moment.

5.9.4 FINISH SPDU

This SPDU initiates orderly release of the session connection.

Table 5.15 Parameters of the FINISH SPDU

SI: 9				
Parameter Group	PGI	Parameter	PI	Length (bytes)
		Transport disconnect	17	1
User data	193			512 max

Transport disconnect
indicates whether or not the transport connection is to be kept. It has the same encoding and default as in the REFUSE SPDU.

User data
contains transparent data supplied by the SS-user.

5.9.5 DISCONNECT SPDU

This SPDU signals the orderly release of the session connection.

Table 5.16 Parameters of the DISCONNECT SPDU

SI: 10				
Parameter Group	PGI	Parameter	PI	Length (bytes)
User data	193			512 max

User data
contains transparent data supplied by the SS-user.

5.9.6 ABORT SPDU

This SPDU is used to reject a session connection establishment attempt, or to cause abnormal release of a session connection at any time. It may also be used to release the session connection when a protocol error is detected.

Table 5.17 Parameters of the ABORT SPDU

SI: 25				
Parameter Group	PGI	Parameter	PI	Length (bytes)
		Transport disconnect	17	1
		Reflect parameter values	49	9 max
User data	193			9 max

Transport disconnect
indicates whether or not the transport connection is to be kept, together with an optional reason code specifying the reason for the abort. The encoding for this field is:

a) bit 1 = 0 : transport connection is to be kept;
b) bit 1 = 1 : transport connect is to be released;
c) bit 2 = 1 : user abort;
d) bit 3 = 1 : provider abort: protocol error;
e) bit 4 = 1 : provider abort: no reason.

Bits 5 to 8 are reserved.

Reflect parameter values
may only be present if the Transport Disconnect parameter indicates "protocol error". It contains an implementation defined value and semantics. This parameter is never used by the X.400 SPM.

User data
may only be present if the Transport Disconnect parameter indicates "user abort" and contains transparent data supplied by the SS-user.

5.9.7 ABORT ACCEPT SPDU

This SPDU is used to return a confirmation to the ABORT SPDU if the transport connection is to be kept. The SPM, as a local implementation decision, may send this SPDU in response to an ABORT SPDU even if the transport connection is not to be kept. This option is not implemented by the X.400 SPM.

This SPDU contains no parameters. Its SI field contains the value 26.

5.9.8 DATA TRANSFER SPDU

This SPDU is used to transfer normal SSDUs.

A S-DATA.request from the SS-user results in a single DATA TRANSFER SPDU unless segmenting has been selected. In this case, an ordered sequence of DATA TRANSFER SPDUs will be sent, each containing an SSDU segment, until the complete SSDU has been transferred. All DATA TRANSFER SPDUs, except the last in a sequence greater than one, must have user information.

A valid incoming DATA TRANSFER SPDU results in an S-DATA.indication to the SS-user unless segmenting has been selected. In this case, a valid incoming DATA TRANSFER SPDU, which indicates end of SSDU, results in an S-DATA.indication to pass the entire, reassembled SSDU to the SS-user.

Where segmenting has been selected and an incomplete SSDU is outstanding, the receipt of:
EXCEPTION DATA SPDU,
ACTIVITY INTERRUPT SPDU,
ACTIVITY DISCARD SPDU, or
ABORT SPDU
has a destructive effect on the entire SSDU. DATA TRANSFER SPDUs which have already been received are discarded and the remaining SPDUs will not be received.

Table 5.18 Parameters of the DATA TRANSFER SPDU

SI: 1				
Parameter Group	PGI	Parameter	PI	Length (bytes)
		Enclosure item	25	1
User information field				unlimited

Enclosure item
is only present if segmenting has been selected. It indicates whether the SPDU contains the beginning, middle or end of the SSDU. The encoding for this field is:

a) bit 1 = 1 : beginning of SSDU;
   bit 1 = 0 : not beginning of SSDU;
b) bit 2 = 1 : end of SSDU;
   bit 2 = 0 : not end of SSDU.

Bits 3 to 8 are reserved. The default value for this field is: bit 1 = 1 and bit 2 = 1, i.e., beginning and end of SSDU (this is always the case if segmenting has not been selected).

User information field
contains a complete or partial normal SSDU. A complete SSDU of unlimited length is carried when segmenting has not been selected. When segmenting has been selected, the size of this field is limited by the maximum TSDU size. This field must be present if the Enclosure Item parameter is not present (i.e., no segmenting has been selected) or has bit 2 = 0 (i.e., not end of SSDU).

5.9.9 GIVE TOKENS SPDU

This SPDU is used to:

a) introduce a concatenated sequence of SPDUs; and/or
b) cause assignment of currently changed tokens to be changed.

Since the X.400 SPM does not provide the Give Tokens service or use extended concatenation, it only uses this SPDU to introduce a basic concatenation of SPDUs. For this

use, this SPDU does not contain any parameters. Its SI field contains the value 1.

5.9.10 PLEASE TOKENS SPDU

This SPDU is used to:

a) introduce a concatenated sequence of SPDUs; and/or
b) request that the token assignments be changed.

When the X.400 SPM uses this SPDU to introduce a basic concatenation of SPDUs, it does not contain any parameter fields. In this case, it does not achieve any Please Tokens function.

Table 5.19 Parameters of the PLEASE TOKENS SPDU

SI: 2				
Parameter Group	PGI	Parameter	PI	Length (bytes)
		Token item	16	1
User data	193			512 max

Token item
indicates which tokens are being requested by the sending SS-user. The encoding for this field is:

a) bit 7 = 1 : release token;
b) bit 5 = 1 : major/activity token;
c) bit 3 = 1 : synchronize-minor token;
d) bit 1 = 1 : release token.

Bits corresponding to tokens which are not available are ignored. The X.400 SPM will therefore always ignore bit 1 since the release token is never available. If this field

is present, at least one bit corresponding to an available token must be set to one.

User data
contains transparent data supplied by the SS-user. It may only be present if the Token Item parameter is present.

5.9.11 GIVE TOKENS CONFIRM SPDU

This SPDU is used to change the assignment of all the currently assigned tokens.

This SPDU contains no parameters. Its SI field contains the value 21.

5.9.12 GIVE TOKENS ACK SPDU

This SPDU is used to acknowledge receipt of a GIVE TOKENS CONFIRM SPDU.

This SPDU contains no parameters. Its SI field contains the value 22.

5.9.13 MINOR SYNC POINT SPDU

This SPDU is used to define a minor synchronization point. A confirmation may be returned by the receiver but is not required by the SPM. All acknowledgement rules are defined by the SS-users. In particular, whether confirmation is requested or not is transparent to the SPM.

Table 5.20 Parameters of the MINOR SYNC POINT SPDU

SI: 49				
Parameter Group	PGI	Parameter	PI	Length (bytes)
		Sync type item	15	1
		Serial number	42	6 max
User data	193			512 max

Sync type item
indicates that explicit confirmation is not required. The encoding for this field is:

bit 1 = 1 : explicit confirmation not required.

Bits 2 to 8 are reserved. This field is absent if an explicit confirmation is required.

Serial number
is the serial number of the minor synchronization point. It has the same encoding as in the CONNECT SPDU.

User data
contains transparent data supplied by the SS-user.

5.9.14 MINOR SYNC ACK SPDU

This SPDU is used to return a confirmation to minor synchronization points.

Table 5.21 Parameters of the MINOR SYNC ACK SPDU

SI: 50				
Parameter Group	PGI	Parameter	PI	Length (bytes)
		Serial number	42	6 max
		User data	46	512 max

Serial number
is the serial number of the minor synchronization point being confirmed. It has the same encoding as in the CONNECT SPDU.

User data
contains transparent data supplied by the SS-user.

5.9.15 EXCEPTION DATA SPDU

This SPDU is used to put the SPM into an error state following a user exception report.

Table 5.22 Parameters of the EXCEPTION DATA SPDU

SI: 48				
Parameter Group	PGI	Parameter	PI	Length (bytes)
		Reason code	50	1
User data	193			512 max

Reason code
indicates the reason for the user exception report. Its value may be one of:

a) 0 : No specific reason;
b) 1 : User receiving ability jeopardized;
c) 3 : User sequence error;
d) 5 : Local SS-user error;
e) 6 : Unrecoverable procedural error;
f) 128 : Demand data token.

All other values are reserved.

User data
contains transparent data supplied by the SS-user.

5.9.16 ACTIVITY START SPDU

This SPDU is used to indicate the beginning of an activity.

Table 5.23 Parameters of the ACTIVITY START SPDU

SI: 45				
Parameter Group	PGI	Parameter	PI	Length (bytes)
		Activity identifier	41	6 max
User data	193			512 max

Activity identifier
is as defined by the sending SS-user.

User data
contains transparent data supplied by the SS-user.

5.9.17 ACTIVITY RESUME SPDU

This SPDU is used to indicate the resumption of a previously interrupted activity.

Table 5.24 Parameters of the ACTIVITY RESUME SPDU

SI: 29				
Parameter Group	PGI	Parameter	PI	Length (bytes)
Linking information	33	Called SS-user reference	9	64 max
		Calling SS-user reference	10	64 max
		Common reference	11	64 max
		Additional reference information	12	4 max
		Old activity identifier	41	6 max
		Serial number	42	6 max
		New activity identifier	41	6 max
User data	193			512 max

Called SS-user reference
Calling SS-user reference
Common reference
Additional reference information
Old activity identifier
are as defined by the sending SS-user.

Serial number
indicates the first synchronization point serial number to be used minus one. It has the same encoding as in CONNECT SPDU.

New activity identifier
is as defined by the sending SS-user.

User data
contains transparent data supplied by the SS-user.

5.9.18 ACTIVITY INTERRUPT SPDU

This SPDU is used to indicate the interruption of the current activity.

Table 5.25 Parameters of the ACTIVITY INTERRUPT SPDU

SI: 25				
Parameter Group	PGI	Parameter	PI	Length (bytes)
		Reason code	50	1

Reason code
indicates the reason for the activity interrupt. Its value may be one of:

a) 0 : No specific reason;
b) 1 : User receiving ability jeopardized;
c) 3 : User sequence error;
d) 5 : Local SS-user error;
e) 6 : Unrecoverable procedural error;
f) 128 : Demand data token.

All other values are reserved. The default value for this field is 0.

5.9.19 ACTIVITY INTERRUPT ACK SPDU

This SPDU is used to notify the sender of an ACTIVITY INTERRUPT SPDU of the completion of the activity interruption.

This SPDU contains no parameters. Its SI field contains the value 26.

5.9.20 ACTIVITY DISCARD SPDU

This SPDU is used to indicate the cancellation of the current activity.

Table 5.26 Parameters of the ACTIVITY DISCARD SPDU

SI: 57				
Parameter Group	PGI	Parameter	PI	Length (bytes)
		Reason code	50	1

Reason code
indicates the reason for the activity discard. Its value may be one of:

a) 0 : No specific reason;
b) 1 : User receiving ability jeopardized;
c) 3 : User sequence error;
d) 5 : Local SS-user error;
e) 6 : Unrecoverable procedural error;
f) 128 : Demand data token.

All other values are reserved. The default value for this field is 0.

5.9.21 ACTIVITY DISCARD ACK SPDU

This SPDU is used to notify the sender of an ACTIVITY DISCARD SPDU of the completion of the activity cancellation.

This SPDU contains no parameters. Its SI field contains the value 58.

5.9.22 ACTIVITY END SPDU

This SPDU is used to indicate the normal termination of the current activity.

Table 5.27 Parameters of the ACTIVITY END SPDU

SI: 41				
Parameter Group	PGI	Parameter	PI	Length (bytes)
		Serial number	42	6 max
User data	193			512 max

Serial number
indicates the serial number of this implied major synchronization point, and is set by the SPM to the next serial number to be used. It has the same encoding as in the CONNECT SPDU.

User data
contains transparent data supplied by the SS-user.

5.9.23 ACTIVITY END ACK SPDU

This SPDU is used to return a confirmation to an ACTIVITY END SPDU.

Table 5.28 Parameters of the ACTIVITY END ACK SPDU

SI: 42				
Parameter Group	PGI	Parameter	PI	Length (bytes)
		Serial number	42	6 max
User data	193			512 max

Serial number
indicates the serial number of the implicit major synchronization point being confirmed, and is equal to the next serial number to be used minus 1. It has the same encoding as in the CONNECT SPDU.

User data
contains transparent data supplied by the SS-user.

6. A FORMAL DESCRIPTION OF THE SESSION LAYER FOR X.400

This section presents a formal description of the session layer for X.400, using the Formal Description Technique (FDT) Estelle. This formal description is based on the session service definition of section 4 and the session protocol specification of section 5.

This section starts with an introduction to FDTs which shows why formal descriptions of communication systems are important. Two FDTs which are currently widely used are introduced. Next, a general overview of the FDT Estelle is presented. This includes a brief discussion of the main features and strengths of the language and explains why Estelle was chosen for this project. This is followed by a presentation of the Estelle specification text and a detailed description of its structure and functionality. Finally, the major coding features of this specification are described.

6.1 Introduction to FDTs

6.1.1 The need for FDTs

OSI layer services and protocols must be described for many purposes in the layer development process. These descriptions must be both easy to understand and precise - goals which often conflict.

The problems of designing and describing distributed, concurrent, information processing systems (such as the OSI layers) are much more difficult than those of classical (sequential) systems. These difficulties arise from the inherent complexity of distributed systems, in which several independent, sequential components may cooperate and execute in parallel. These problems are compounded when the design and implementation of communication software is considered. Here, compatibility

must be ensured between different, sometimes heterogeneous, system components which are often implemented by different groups of people in different organizations. These problems indicate the vital need for precise, unambiguous system descriptions.

Traditionally, OSI layer services and protocols are described using a combination of natural language descriptions, informal walk-throughs and state tables. Although such informal methods have been largely successful in layer development, they are by themselves inadequate, having yielded errors or unexpected and undesirable behaviour in most protocols. Their use gives the illusion of being easily understood, but leads to lengthy, informal descriptions which often contain ambiguities and are difficult to check for completeness, consistency and correctness.

FDTs, on the other hand, provide a much better format for describing distributed systems. Their advantages stem from their following fundamental characteristics:

a) FDTs are well-defined. They are based on formally-defined abstract modelling techniques, mathematics, syntax and semantics.

b) FDTs are well-structured. They structure a system description in a manner that is meaningful and intuitively pleasing. This increases the readability, understandability, analyzability and maintainability of system descriptions.

c) FDTs are abstract. They are completely independent of implementation methods, so that the technique itself does not constrain implementors. In addition, they provide abstraction from irrelevant details by including only the essential requirements that the system must satisfy and omitting the unessential.

d) FDTs are expressive. They are able to describe both the service definitions and protocol specifications of the OSI layers.

FDTs, therefore, produce system descriptions which are complete, consistent, precise, concise and unambiguous.

6.1.2 Applications of FDTs

Formal system descriptions have many applications and advantages in all three stages of the system development life-cycle:

the System Design stage,
the System Validation stage, and
the System Implementation stage.

The system design stage:

Formal descriptions of early system designs serve as an accurate reference for cooperation among designers of different system parts, and accustom designers to more disciplined approaches. More importantly, they enable ambiguities, inconsistencies and problems, which would otherwise have remained unnoticed, to be detected and removed from the design. Furthermore, they greatly improve the presentation, readability and clarity of system descriptions.

The system validation stage:

The mathematical, syntactic and semantic formality of a formal description lends itself to formal analysis by automated, computer-based tools.

Such analysis may range from simple syntax checking to full syntax and semantic checking. More sophisticated analysis may provide system *validation*, which aims to show that a system satisfies its design specifications and operates as desired. Validation is important in all stages of system development and may be realized by the application of techniques like system simulation, verification, performance analysis, conformance testing and testing of the final implementation. All these techniques, except the last one, depend on a formal system description.

System *simulation* may be performed by applying some type of emulator or text-interpreter tool to the formal system description.

While simulation and testing only validate the system for certain test situations, *verification* allows, in principle, the consideration of all possible situations that the system may encounter during actual operation. Verification checks both general and specific system properties. General properties include the absence of deadlock and infinite loops, completeness and proper progress and termination. Specific properties relate to the particular actions to be performed by the system. Unfortunately, verification tools often consume huge amounts of CPU time, while specification state explosion often makes full coverage impossible.

Performance analysis allows performance predictions to be made based on the formal system description before any implementation is available. Performance issues include throughput, delays, error rates, etc.

While the system design specification need only be verified once, *conformance testing* entails testing the equivalence of two or more specifications, or the testing

of different implementations for compliance with the system design specification.

Clearly, most validation work is performed on the formal system description before an implementation is available. It is a well-known fact the cost of error correction increases quickly during the system development life-cycle. Therefore, it is cost-effective to detect and correct errors during the initial stages of system development.

The system implementation stage:

A formal system description may serve as a basis for system implementation. This may take the form of a hand translation from the formal description to some target high-level programming language such as Pascal, Ada, C or CHILL [12] (the CCITT High Level Language - designed specifically for communication system implementations). A better translation method is to use a computer-based, compiler-like tool which automatically derives an implementation (or parts thereof) from a formal description. This method is both faster and more reliable than hand translation.

6.1.3 Current FDTs

Generally, FDTs apply many different techniques to the problem of system description. These are well described in the literature [13], [14]. Broadly, these techniques may be classified into three main categories: pure state transition models, programming languages and hybrid combinations of the first two. Although all three have their advantages and disadvantages, the hybrid model has proved to be the most successful because it combines the advantages of the first two.

There are currently two FDTs (both of the hybrid type) which are in popular, widespread use: CCITT's *Specification and Description Language* (SDL) and ISO's *Estelle*.

SDL is intended for use as a common language for the CCITT standards (Recommendations) and is defined in the CCITT Z.100 series of SDL Recommendations [15], [16]. It has both a graphical and a text-based syntax. Current support tools for SDL include a graphical editor capable of handling SDL's "process diagrams" [17].

Estelle is a text-based FDT developed by ISO in collaboration with the CCITT [6]. It is a language for specifying distributed systems with a particular application in mind, namely that of the OSI layer protocols and services.

6.2 An overview of the FDT Estelle

The FDT Estelle is defined for specifying distributed, concurrent, information processing systems. In particular, it can be used to describe the layer services and protocols of the OSI architecture.

6.2.1 The major features of Estelle

Estelle is based on an extended finite state transition model. This means that it is a hybrid combination of a pure finite state transition model and a high-level programming language - in this case, *Pascal*. Its small-state transition model captures the main features of the system (e.g. connection establishments, data transfers, disconnects, etc.), which is then augmented with additional "context" variables (e.g. serial numbers) and processing routines for each state. Actions to be taken are determined by using parameters from the inputs and values of the context variables according to the processing routines for each state.

A distributed system is viewed in Estelle as a structure of communicating components called *module instances*, or *tasks*. Each task has a number of input/output ports called *interaction points*. The internal behaviour of a task is described in terms of a nondeterministic communicating state machine whose actions are given as Pascal statements (with some restrictions and extensions). Tasks may be nested to form a hierarchical task structure. Furthermore, a communication structure exists between the tasks, which are linked via their interaction points. Tasks communicate with each other and with their environment by sending and receiving *interactions* via their interaction points. Parent tasks have the means to create and destroy instances of their child tasks and their communication links.

The detailed semantics and syntax of Estelle are not described here and a thorough familiarity with them are assumed. For more information, the reader is referred to the Estelle definition [6] and to two good tutorials on the features and facilities of Estelle in [18] and [19].

6.2.2 Estelle support tools

Although Estelle is a relatively new language, a considerable international effort is being made in developing specifications in Estelle and in designing support tools for it. In particular, such tools are being developed within two projects of the European Esprit program: SEDOS (Software Environment for Design Open Systems - November 1984 to October 1987) and SEDOS-ESTELLE-DEMONSTRATOR (June 1986 to May 1989). Prototype tools such as an editor, compiler, interpreter and a simulator/debugger will be the outcome of the SEDOS project, while an Estelle Work Station will be developed within the SEDOS-ESTELLE-DEMONSTRATOR project. Similar efforts in developing Estelle tools are being made in the U.S.A., Canada and Japan [18].

6.2.3 Motivation for selecting Estelle

Estelle has been selected as the FDT for this project for a number of reasons:

a) Estelle closely resembles the Pascal programming language, which is familiar to most computer scientists and many engineers.

b) Estelle has a text-based syntax (as opposed to a graphical syntax) which simplifies the mapping from conventional (text-based) layer descriptions into Estelle.

c) One of the strengths of an Estelle specification is that it indicates, by virtue of its Pascal-like constructs, how an implementation may be derived from it. This will be important in section 7, which shows how the session layer for X.400 may be implemented on a real system.

d) Many computer-based support tools for Estelle should be readily available in the near future, as described is subsection 6.2.2.

The Estelle version used in this project is the ISO Second Draft Proposal (DP9074), 1986 [6]. This was the only version available to the author at the time of writing.

6.3 The Estelle specification structure

The complete Estelle specification of the session layer for X.400 is listed in APPENDIX C. The reader is referred to this well-commented listing for any detailed information concerning the specification. This subsection supplements this specification by describing its structure and functionality.

This subsection starts by presenting an OSI model of the session layer for X.400 on which the specification is based. This is followed by a structure diagram of the specification and a detailed description of each module.

6.3.1 A model of the session layer for X.400

A formal description of any system must be based on an abstract model of that system. In the case of the session layer, the most effective way to model it is to model only one *type* of each of its simplest, elementary, structural components. *Instances* of these types may then be created and configured to represent any real session layer at any instant of its life-time.

Figure 6.1 depicts such an elementary model of the session layer for X.400, in terms of OSI abstract modelling concepts.

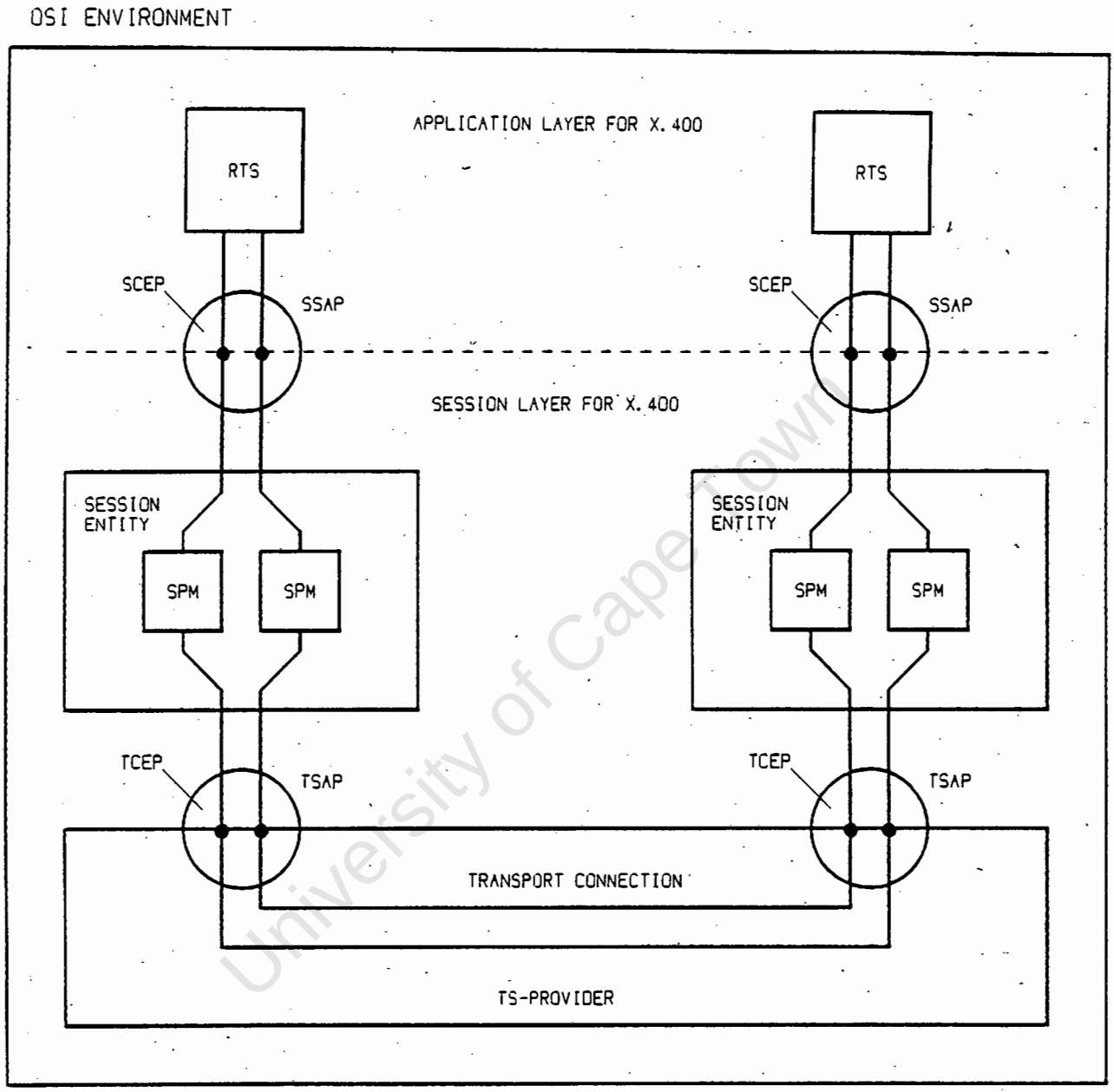

Figure 6.1 An OSI model of the session layer for X.400

The session layer of Figure 6.1 provides session connections for two correspondent RTSs. Each RTS is supported by a single session entity through a single SSAP.

Because it is assumed that each session entity supports only one SSAP (and therefore one RTS), and because of the one-to-one mapping between SSAP and TSAP addresses for X.400, each session entity requires only one TSAP to the TS-provider.

A session entity may support several simultaneous session connections at its SSAP, so its SSAP has several SCEPs. Because of the one-to-one mapping between session and transport connections, several simultaneous session connections require an equal number of simultaneous transport connections. Therefore, the session entity's TSAP has a number of TCEPs, equal to the number of SCEPs.

Within a session entity, an SPM is responsible for supporting each SCEP and its corresponding TCEP. The session entity therefore nests as many SPMs as it has SCEP/TCEP pairs.

This model does not show the position of the presentation layer because, as shown in section 3, it intervenes "transparently" between the X.400 application layer and the session layer. Neither does this model consider individual open systems. Instead, it assumes that an RTS and its supporting session entity reside in the same open system, and that the TS-provider (representing all layers below the session layer) links all open systems of the OSI environment.

6.3.2 The Estelle specification structure

Figure 6.2 depicts a structure diagram of the Estelle specification. This diagram is the Estelle equivalent of the OSI model of Figure 6.1. Because of this equivalence, the Estelle specification structure is familiar and intuitively pleasing.

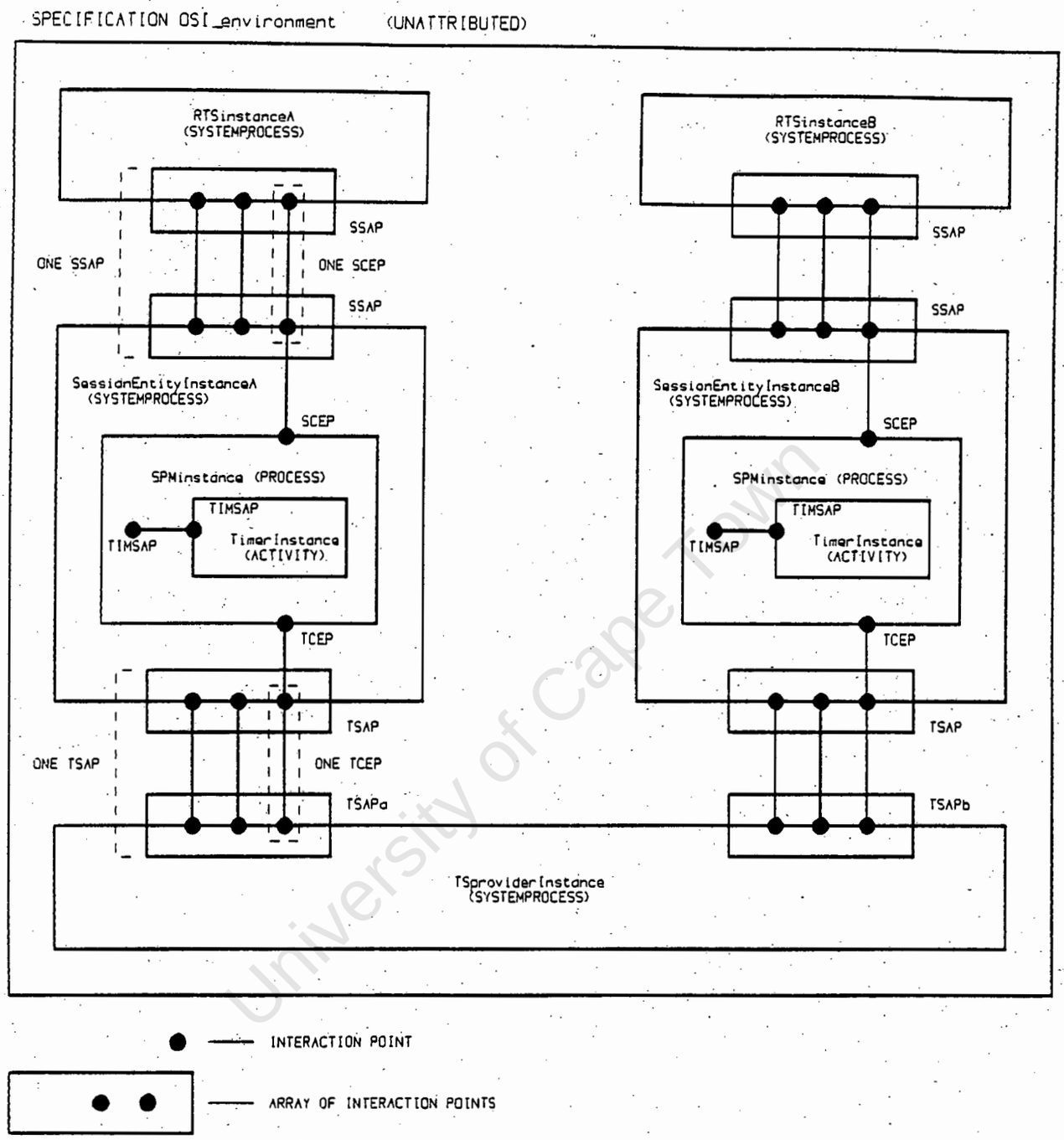

Figure 6.2 The Estelle specification structure diagram

Each session entity task is connected to the RTS task it supports through one SSAP. An SSAP is modelled by a combination of three Estelle constructs:

a) An array of external interaction points of the RTS task. These are of SSAP-type and assume the role of SS-user.

b) An array of external interaction points of the session entity task. These are also of SSAP-type but assume the opposite role to those of the RTS task, namely that of SS-provider. This array has the same number of elements as the RTS task's array.

c) Static connections between each corresponding pair of elements of these two arrays.

The identifier of the session entity task's SSAP array is equivalent to its SSAP selector. Although SSAP selectors are not required by X.400, they are pointed out here purely for interest's sake.

Each connected pair of SSAP array elements represents a SCEP. The (identical) array indices of the connected pair is equivalent to the SCEP identifier.

Each session entity task is connected to the TS-provider task through a single TSAP. A TSAP is modelled by a combination of three Estelle constructs:

a) An array of external interaction points of the session entity task. These are of TSAP-type and assume the role of TS-user.

b) An array of external interaction points of the TS-provider task. These are also of TSAP-type but assume the opposite role to those of the session entity task, namely that of TS-provider. This array has the same number of elements as the session entity task's array.

c) Static connections between each corresponding pair of elements of these two arrays.

The identifier of the TS-provider task's TSAP array is equivalent to the TSAP address. Because X.400 specifies a one-to-one mapping between SSAP and TSAP addresses, the TSAP address is equivalent to the SSAP address. The TS-provider task's two TSAPs (TSAPa and TSAPb) therefore address two different SSAPs and therefore two different RTS tasks.

Each connected pair of TSAP array elements represents a TCEP. The (identical) array indices of the connected pair is equivalent to the TCEP identifier.

The specification module is unattributed and therefore inactive. This has three important consequences:

a) It defines a static configuration of child tasks with a static communication structure between them. Although the OSI Reference Model does not specify that the layer entity instances and the relationships between them are necessarily static, this assumption is valid for the purposes of this specification.

b) Its child tasks must be *system* tasks, so the SYSTEMPROCESS attribute has been chosen for each. The reason for this choice will become clear when the session entity module is described.

c) Its child tasks execute fully asynchronously with respect to each other. This is in full agreement with the OSI Reference Model.

6.3.4 The session entity module

This module represents a session entity. It creates several, identical child tasks: the SPMs.

Each SPM task has two external interaction points:

a) One is of SSAP-type and assumes the role of SS-provider. It represents the SCEP which the SPM task supports and is intended for attachment to one of the session entity task's SCEPs within its SSAP array.

b) The other is of TSAP-type and assumes the role of TS-user. It represents the TCEP which the SPM task uses and is intended for attachment to one of the session entity task's TCEPs within its TSAP array.

The session entity task creates as many SPM tasks as it has SCEP/TCEP pairs. It then permanently attaches each between one of its SCEP/TCEP pairs.

The session entity module (and all other child modules of the specification module) has the SYSTEMPROCESS attribute and may therefore be active. This has three important consequences:

a) It defines a dynamic configuration of child tasks with a dynamic communication structure between them. This means that the session entity task may create and destroy SPM tasks, and any links with them, as required. This is in full agreement with the OSI Reference Model.

b) Its child tasks must have either the PROCESS or the ACTIVIY attribute. The PROCESS attribute has arbitrarily been chosen for each SPM task. The reason for this choice will become clear when the SPM module is described.

c) Its child tasks may execute in parallel if they are not in parent/child conflict, i.e., parallel but synchronized by the parent/child priority principle. This means that the SPM tasks within a session entity task execute in parallel, but execution of the session entity task takes priority over them. This is in full agreement with the OSI Reference Model.

If the session entity module had been assigned the SYSTEMACTIVITY attribute, this would restrict all its descendent tasks to the ACTIVITY attribute, which would restrict them to the nondeterministic sequential execution mode (while preserving the parent/child priority principle). This would conflict with the OSI Reference Model which assumes that all its entities execute in parallel.

The session entity task's SCEPs, SPM tasks and TCEPs represent its resources. Naturally, it must utilize and manage these resources as efficiently as possible in order to avoid unnecessary congestion resulting from the allocation of resources to inactive connections. This calls for some resource management algorithm which creates SPM tasks, attaches and detaches them to or from SCEPs and TCEPs and then releases them as required.

The ideal algorithm, which would enable the session entity task to use as little of its available resources as possible in supporting the required session connections, would operate as follows:

a) An active session connection supported by an active transport connection requires that one SPM task is attached between the corresponding SCEP/TCEP pair. The session entity task monitors the SPM task's state transitions through, say, an exported variable of the SPM task.

b) If the session entity task detects that the SPM task has entered state STA01 (idle, no transport connection) it detaches the SPM task from the TCEP and the SCEP (if there is such an attachment) and releases the SPM task. This frees all the resources associated with those connections.

c) If the session entity task detects that the SPM task has entered state STA01C (idle, active transport connection) it detaches the SPM task from the SCEP while retaining the SPM task and its TCEP attachment. The SCEP is thus freed while the SPM task supports a reusable transport connection. In this case, the session entity task would know the characteristics (QOTS, called TSAP address) of this reusable transport connection through, say, exported variables of the SPM task.

d) If the session entity task detects an incoming S-CONNECT.request on one of its free SCEPs, it first looks for an SPM task supporting a suitable (in terms of QOTS and called TSAP address), reusable transport connection. If it finds one, it attaches the SCEP to this SPM task. If it does not find one, it looks for a free TCEP. If it finds one, it creates an SPM task and attaches it to the SCEP and the TCEP. If it does not find one (i.e., all its resources have been allocated), it rejects the incoming S-CONNECT.request by issuing an S-CONNECT.confirm (reject) on the same SCEP, citing "SS-provider congestion" as the reason for rejection.

e) If the session entity task detects an incoming S-CONNECT.indication on the SCEP of one of its SPM tasks supporting a reusable or active transport connection, it attaches this SCEP to one of its own, free SCEPs. In this case it will always find a free SCEP because the existence of a reusable transport connection implies the existence of a free SCEP.

f) If the session entity task detects a T-CONNECT.indication on one of its free TCEPs, it creates an SPM task and attaches it to the TCEP.

Although ideal, this resource allocation algorithm is not implemented by the session entity module of this Estelle specification. There are two reasons for this: Firstly, this algorithm is too complex in terms of Estelle coding. If coded in Estelle, it would be long, clumsy, and would detract from the relative simplicity, clarity and neatness of the specification. Secondly, neither the OSI Reference Model nor the OSI Session Layer specifications make any mention of session entity resource allocation (and optimization) algorithms like this one. Clearly, these are implementation-dependent issues. What might be necessary (or efficient) optimization for one implementation may not be so for another. It is best to leave such issues up to individual implementations by not constraining them at this stage.

The specification's session entity module therefore employs the simplest possible method for allocating its resources. It creates as many SPM tasks as it has SCEP/TCEP pairs and permanently attaches each between one of its SCEP/TCEP pairs. Each SCEP is therefore permanently supported by an SPM task, which, in turn, has a TCEP permanently at its disposal. The advantage of this method is that the session entity task does not have to monitor the SPM tasks, neither does it have to implement any part

of the session protocol, thereby allowing all the session protocol functionality to be concentrated in the SPM task. Recall that, in the previous method, the session entity had to respond to certain incoming session and transport primitives while issuing others.

The major disadvantage of this method is that it does not allow the SPM tasks to reuse their transport connections efficiently. If an SPM retained its transport connection following a session connection release and it then received another S-CONNECT.request which had transport connection requirements (QOTS and called TSAP address) which are different to those of its reusable transport connection, it would have to reject the connection attempt. Alternatively, it could release its transport connection and establish the required one. Clearly, neither of these actions represent desirable behaviour or effective resource utilization, and defeat the object of reusable transport connections.

Another drawback of this method is that, if translated directly into a real implementation, it would result in gross wastage of system resources such as memory and CPU time because all possible SPM tasks exist permanently, whether active or not. However, this does not matter in a formal description because it assumes the existence of unlimited system resources.

6.3.5 The SPM module

This module represents a single SPM which performs the session protocol for X.400 as specified in section 5. It creates a single child task which represents the session protocol timer, TIM.

The timer task is accessed through one external interaction point. The SPM task has one internal

interaction point which is permanently connected to the timer task's external interaction point.

The SPM module has the PROCESS attribute and may therefore be active. This has three important consequences:

a) It defines a dynamic configuration of child tasks and a dynamic internal communication structure. This means that the SPM task may create and destroy its timer task, and its link with it, as required.

b) Its child tasks must have either the PROCESS or the ACTIVITY attribute. The ACTIVITY attribute has arbitrarily been chosen for the timer task. The reason for this choice will become clear when the timer task is described.

c) Its child tasks may execute in parallel if they are not in parent/child conflict, i.e., parallel but synchronized by the parent/child priority principle. However, because the SPM task nests only one child task, any attribute that it may have has no effect on the execution of its child. This implies that the SPM task may have been assigned either the PROCESS or the ACTIVITY attribute without affecting its internal operation. The execution of its child is therefore subject only to the parent/child priority principle. This means that execution of the SPM task has priority over execution of the timer task, as required.

6.3.6 The timer module

This module represents the session protocol timer, TIM. It does not nest any child modules.

The timer module has the ACTIVITY attribute, and may therefore be active. Because it nests no child modules, its attribute has no effect on its internal operation and it may just as well have been assigned the PROCESS attribute.

The internal operation of the timer module is not specified by the session protocol, so its design and operation is presented here:

Figure 6.3 depicts a state transition diagram of the timer module's operation.

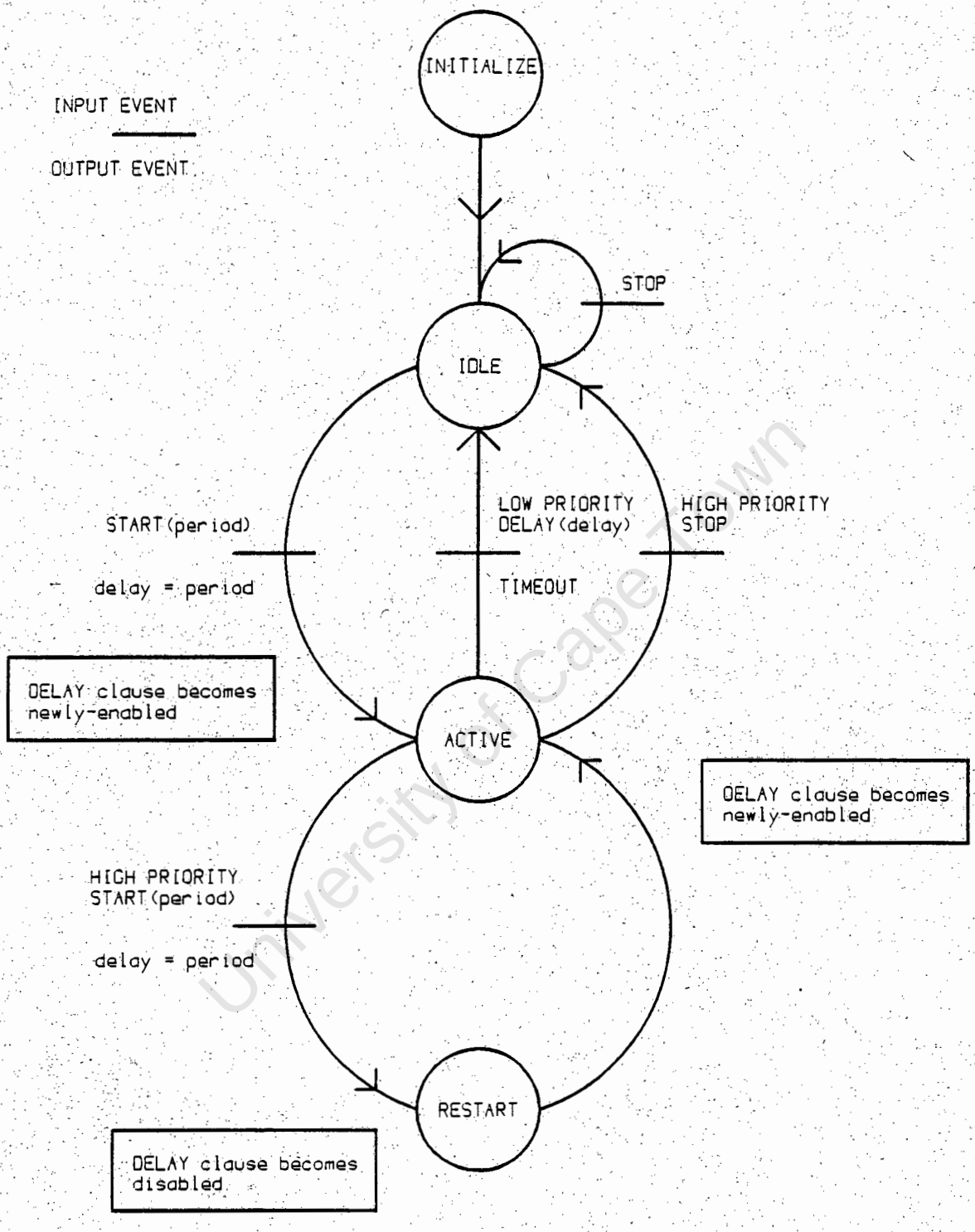

Figure 6.3 State transition diagram for the timer module

The timer task enters the IDLE state when it is initialized. Here it awaits a START interaction from the user, ignoring any other interactions.

When it receives a START interaction, it loads its **delay** variable with the time period specified by the START interaction and enters the ACTIVE state.

On entering the ACTIVE state, the DELAY clause becomes newly-enabled. This causes the DELAY clause's timer to load the value in **delay** and start its countdown. The DELAY clause timer countdown continues as long as the DELAY clause remains enabled, i.e., as long as the timer task remains in the ACTIVE state. One of three transitions may now occur:

1) The DELAY clause timer expires. The timer task sends the TIMEOUT interaction to the user and enters the IDLE state.

2) The timer task receives the STOP interaction from the user. The timer task enters the IDLE state. This disables the DELAY clause, halting its timer. Because it is more important for the user to cancel the timer than it is for the timer to expire, this transition has a higher priority than that of 1), ensuring that it will be selected should both become fireable simultaneously.

3) The timer task receives the START interaction from the user. This means that the DELAY clause timer must be reloaded with the new time period specified by the START interaction and its countdown restarted.

   The **delay** variable is loaded with the new time period. This action is not sufficient to restart the DELAY clause timer because the DELAY clause timer is started only when the DELAY clause becomes newly-enabled. To

solve this problem, the timer task first enters the RESTART state. This disables the DELAY clause, halting its timer. From the RESTART state, the timer task executes a spontaneous transition back to the ACTIVE state. The DELAY clause is now newly-enabled again, causing its timer to be restarted with the new value in **delay**.

Because it is more important for the user to restart the timer than it is for the timer to expire, this transition has a higher priority than that of 1), ensuring that it will be selected should both become fireable simultaneously.

Note that the actual time period with which this timer is loaded is related to the Quality of Service provided by the session connection it serves. It is therefore a local, implementation-dependent matter.

6.4 The Estelle specification coding features

The Estelle specification has been designed and written so as to be as logical, clear, concise, precise and readable as possible. To this end, it is thoroughly commented, making it unnecessary to give a step-by-step description of its instructions here. Instead, this subsection discusses the most important coding features of the specification.

6.4.1 General coding features

The following discussion concerns general features which are applicable to all parts of the specification.

1) The specification is partitioned strictly along the lines of the various specification "parts" described in the Estelle language definition [6]. This is a useful aid in following the structure and development of the specification.

2) The specification was written so as to resemble the narrative session service and protocol definitions of sections 4 and 5 as closely as possible. To achieve this, the terminology, naming conventions, identifiers and abbreviations used for constants, variables, functions, procedures, primitives, parameters, tokens, functional units, SPDUs, predicate expressions, specific actions, etc. are as consistent as possible with those established in previous sections of this thesis. This aids in clarifying both the mapping between OSI abstract modelling concepts and equivalent Estelle constructs, and the derivation of the Estelle specification from the service and protocol definitions of sections 4 and 5.

3) Standard coding conventions are used in the specification: constant identifiers contain only capital letters, digits and underscores, while variable identifiers contain at least one small letter.

6.4.2 Specific coding features

The following discussion concerns the coding features of several specific parts of the specification.

1) All the multi-byte data types used as parameters to service primitives and SPDUs are constructed from Pascal records and arrays. These types are often passed as parameters to Pascal functions and procedures.

   If this scheme were to be translated directly into a real implementation, it would result in very inefficient code because of the extensive stack operations required when passing arrays as parameters. To avoid this problem, a real implementation should use pointers to data areas which contain the multi-byte data types. Only one copy of each multi-byte data type need then ever exist, while only its pointer is passed between procedures and functions.

   In a formal description such as this, such a data-typing scheme is quite acceptable because the formal description does not necessarily translate into any particular real implementation. In addition, data-typing in a formal description should be chosen to be as clear and as readable as possible. This case serves to illustrate the point that great care must be taken when translating a formal description into a real implementation to avoid clumsy, inefficient code generation.

2) The SPM module defines several local constants which configure its externally-visible behaviour. These constants are:

VERSION	- session protocol version number;
PROTOCOL	- session protocol options number;
TEXP_LOCAL	- transport expedited data option;
REUSE_TC	- transport connection reuse option;
DEFAULT_QOSS	- default QOSS values;
DEFAULT_QOTS	- default QOTS values;

FU_SUP - supported functional units;
PERIOD - session protocol timer period.

Naturally, the behaviours defined by these constants will be common to all (identical) instances of the SPM module, and therefore common to the Session Entity module as a whole. Therefore, these constants should instead be defined by the Session Entity module where they would form part of the common, external context environment in which the SPM module instances operate. This scheme would be intuitively pleasing because the (common) behaviour of a session entity's session connections is seen as a characteristic of the session entity itself, not of its individual SPMs.

The reason why this specification defines these constants in the SPM module is so as to concentrate all session protocol elements in only one module, thereby increasing the clarity of the specification.

3) The transition-declaration-part of the SPM module performs all the processing of the session protocol for X.400. Its transitions are derived directly from the state tables of APPENDIX B and have been designed to resemble the state table entries as closely as possible.

The transitions do not, however, follow the layout of the state tables exactly. Whereas the state tables are arranged according to the services and phases of the session protocol, the transitions are arranged in numerical initial-state order, from states STA01 to STA713. This layout makes the transitions neater, clearer and more readable.

The basic mapping between state table entry elements and the corresponding Estelle transition elements is as follows:

state table entry element	transition element
initial state	FROM clause
input event	WHEN clause
predicates, primitive & SPDU identification	PROVIDED clause
next state	TO clause
processing actions	compound statement
specific actions	procedure calls
output event	OUTPUT statement

4) The Synchronization Point Serial Number parameter, which is used by a variety of session service primitives and SPDUs, has been defined to be of INTEGER type. This data typing is not entirely accurate because it implies that the serial number parameter may be assigned any positive or negative integer, when, in fact, it is restricted to range from 0 to 999999 only.

The alternative, more accurate typing (from the specification's point of view) would be to create an ordinal, sub-range type for serial numbers which would include only the valid range of serial number values. This approach has been avoided in this specification because it would lead to clumsy translations between ordinal and integer values when performing the many required integer calculations involving serial numbers.

6.4.3 Implementation-dependent issues

The following discussion concerns those specification issues which will affect the characteristics of any real implementation derived from it.

1) The Session Service Definition of CCITT Recommendation X.215 defines the amount of data (the SSDU) that an SS-user may issue with one S-DATA service primitive to be unlimited. In addition, a service primitive is defined to be indivisible and is transferred across the service boundary 'instantaneously'.

   These definitions pose some important considerations for implementations. Firstly, any real implementation will impose effective limitations on SSDU sizes, if only because there is a limit to the amount of data that is actually held on a given computer at any one time. Secondly, it is clear that no real implementation can deliver an arbitrarily large amount of data across an interface in zero time.

   One way to solve these problems is to design the implementation so that an SSDU is transferred in discrete, finite amounts (perhaps to fit in with a buffering policy) and that, therefore, a very large SSDU will be moved across the service boundary in several physical data units. In this case, consistency with the session service definition can only be maintained by careful consideration of the fact that the data primitive has not been transferred until the last physical item of data has been transferred. In fact, there will be some local interaction between the SS-user and the SS-provider concerned with flow control. The nature of this interaction is not precisely defined in the session service definition

because it is impossible to be precise without over-constraining implementations.

Exactly the same general issues described above apply to the transfer of TSDUs in T-DATA primitives across the transport service boundary.

The Estelle specification circumvents these problems by defining two data structures, one for SSDUs and the other for TSDUs, which are (conceptually) large enough to hold arbitrarily large SSDUs and TSDUs. The 'MAXSSDULEN' and 'MAXTSDULEN' constants defined in the Specification module define the maximum sizes of these data structures, and may be assigned arbitrarily large values in accordance with the Estelle language definition. This scheme allows entire, unlimited-length SSDUs and TSDUs to be transferred in single data primitives (Estelle *interactions*), 'instantaneously' across service boundaries.

2) The 'NSCEPS' and 'NTCEPS' constants defined in the Specification module determine how many simultaneous session and transport connections the Session Entity module can support, respectively. Actual values for these parameters will be determined by the requirements of the host system in which a real implementation of the Session Entity module is installed.

3) The 'PERIOD' constant defined in the SPM module represents the time period, in seconds, with which the session protocol timer, TIM, is loaded. Its actual value is related to Quality of Service issues and is therefore implementation-dependent.

4) As shown if sections 4 and 5, the session layer for X.400 uses default values for QOSS and QOTS parameters. The SPM module therefore provides two variables, DEFAULT_QOSS and DEFAULT_QOTS, to hold these values. These variables are treated as constants and are initialized in the initialization-part of the SPM module. The actual values with which they are initialized are obviously implementation-dependent and should only be selected when a real implementation of the SPM (or Session Entity) module is installed in its host system.

The only three members of the DEFAULT_QOSS variable which are initialized to specific values are the following:

a) 'Protection' is set to 'LEVEL_A', indicating that the SPM module provides no data security features. The issue of providing such features will require a substantial amount of further study and is therefore beyond the scope of this thesis.

b) 'ExtendedControl' is set to 'FALSE', indicating that the X.400 SPM does not provide the Expedited Data functional unit nor does it use the Transport Expedited Data Transfer service.

c) 'OptimizedDialogueTransfer' is set to 'FALSE', indicating that the X.400 SPM does not support the SS-provider Extended Concatenation protocol option.

The only member of the DEFAULT_QOTS variable which is initialized with a specific value is 'Protection', which is set to 'LEVEL_A'. This indicates that the X.400 SPM has no data security requirements of the transport layer.

5) CCITT Recommendation X.225 states that the Reflect Parameter Values parameter of the ABORT SPDU has implementation-defined values and semantics. This formal description assigns no values or semantics to this parameter, neither does it recognize any.

6) CCITT Recommendation X.214 states that the TS-user Data parameter of transport service primitives has implementation-defined values and semantics. This formal description assigns no values or semantics to this parameter, neither does it recognize any. Naturally, this argument excludes the TS-user Data parameter of the T-DATA primitives because they contain TSDUs.

7.2 Overview of the X.400 product

This implementation interfaces to an existing X.400 product which provides a complete X.400 application subsystem in the form of software modules for the Message Transfer Agent (MTA), the Reliable Transfer Server (RTS) and the User Agent (UA).

This software is intended for installation in any host system which supports the C programming language in a UNIX environment. It is supplied in source form so that any porting to the host system may be undertaken during installation.

Once the source files have been ported and installed, the executable files are created by running the supplied **makefile** in the X.400 product's root directory. The executable files must then be relocated from their creation directories to the user-specified directories in which they are to run.

The intention is that the MTA and RTS processes run as daemons under the exclusive control of the system administrator. In contrast, each X.400 user on the system gets his own copy of the UA executable file, which he invokes whenever he requires access to the X.400 system.

7.3 Software overview

This subsection provides a general overview of the software associated with this implementation. First, the approach used by the RTS to interface with the host's session and transport layer software is described. This is followed by a statement of the broad requirements for implementing the session layer and interfacing it to the RTS. Next, the extent of this implementation is specified. Lastly, the file structure in which the software resides and the programming conventions used are described for future reference.

Details concerning the software development system itself may be found in APPENDIX D.

7.3.1 Interfacing the RTS to the session layer

In the X.400 MHS Model, the RTS is an X.400 SS-user which interfaces directly with the session layer. Within the session layer, only one session entity is required to support the RTS through one SSAP.

The X.400 product does not treat the RTS and its supporting session entity as two separate processes. Instead, the RTS source code is compiled together with the session entity source code when forming the RTS executable file.

As an aside, note that the collection of session entities of all RTS instances, together with any other session entity instances resident on the host, represents the host's *session subsystem*.

The session entity source code links to the RTS source code at compile-time as an archive-type function library. An archive is generated by the C Development System archive utility, **ar**. The RTS's **makefile** expects this archive **in bas/session.a**.

Evidently, the reason for this archive scheme is to allow the code for a general session entity to be written as groups of functions, each group implementing a different functional unit. These function groups are then installed as separate members in the archive. During compilation, the RTS code then 'extracts' from this archive only those groups of functions associated with the functional units it requires. This avoids compiling unnecessary session entity code, thereby minimizing the size of the executable file. In this implementation, however, the session entity

installed in the archive only provides those functional units required by the RTS, so no real advantage is gained from this scheme.

7.3.2 Interfacing the RTS to the transport layer

Within the transport layer, only one transport entity is required to support a session entity through one TSAP.

The X.400 product does not treat the RTS (with its supporting session entity) and its supporting transport entity as two separate processes. Instead, the RTS source code is compiled together with the transport entity source code when forming the RTS executable file.

The transport entity source code links to the RTS source code at compile-time as an object file. The RTS's **makefile** expects this file in **tp2/transport.o**.

It appears that the reason for using an object file, as opposed to an archive file, for the transport entity code stems from the fact that the transport layer does not provide many optional services to its users. Therefore, all the transport entity code will always be linked to the RTS code, making an archive file unnecessary.

Figure 7.1 illustrates the (conceptual) structure of the RTS executable file.

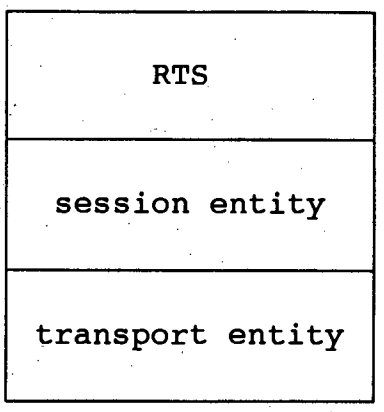

Figure 7.1 Structure of the RTS executable file

This method of combining the RTS, session entity and transport entity source modules into a single executable file has one major advantage: it eliminates the need for elaborate inter-process communication schemes between separate RTS, session entity (or possibly session subsystem) and transport entity (or possibly transport subsystem) processes. This method allows these modules to communicate with each other by means of direct function calls into each other's code.

7.3.3 Implementation requirements

The task of implementing the session entity for the existing RTS entails writing the necessary session entity functions, as specified by the RTS's session interface definition, and then installing them in **bas/session.a**. The RTS's session interface definition is presented in subsection 7.5.

The task of implementing the transport entity for the existing RTS entails writing the transport entity functions, as required by the session entity's transport interface definition, and then installing them in

tp2/transport.o. The session entity's transport interface definition is presented in subsection 7.6.

7.3.4 Extent of implementation

This session entity implementation is partial in the sense that it implements only the following parts of the session protocol:

a) The complete Session Connection Establishment Phase, as defined by State Table B.1 of APPENDIX B, is implemented.

b) For all states associated with a), correct response to aborts is also implemented. These actions are defined in State Table B.8 of APPENDIX B.

c) None of the invalid intersection transitions of State Table B.9 of APPENDIX B are implemented. Instead, invalid intersections are simply ignored.

d) All session protocol timer actions are implemented.

e) Because all the SPDUs associated with actions a) and b) are category 1 SPDUs (i.e., the SPDU is always mapped one-to-one into a TSDU), no Basic Concatenation is implemented.

These restrictions imply that only the following session protocol elements are implemented:

session service primitives:
S-CONNECT.request, S-CONNECT.indication,
S-CONNECT.response, S-CONNECT.confirm,
S-U-ABORT.request, S-U-ABORT.indication,
S-P-ABORT.indication.

transport service primitives:
T-CONNECT.request, T-CONNECT.indication,
T-CONNECT.response, T-CONNECT.confirm,
T-DATA.request, T-DATA.indication,
T-DISCONNECT.request, T-DISCONNECT.indication.

SPM states:
STA01, STA01A, STA01B, STA01C, STA02A, STA08, STA16, STA713.

SPDUs:
CONNECT, ACCEPT, REFUSE, ABORT, ABORT ACCEPT.

specific actions:
1, 2, 3, 4, 5, 11.

predicate conditions:
1, 2.

For testing purposes, a simple transport entity has been implemented to support the session entity. This issue will be discussed in subsection 7.10.1.

7.3.5 The file structure

Figure 7.2 depicts the file structure in which all files specifically referred to in the rest of this section reside. The APPENDIX in which a listing of a file may be found is indicated in **bold** font to the right of the filename.

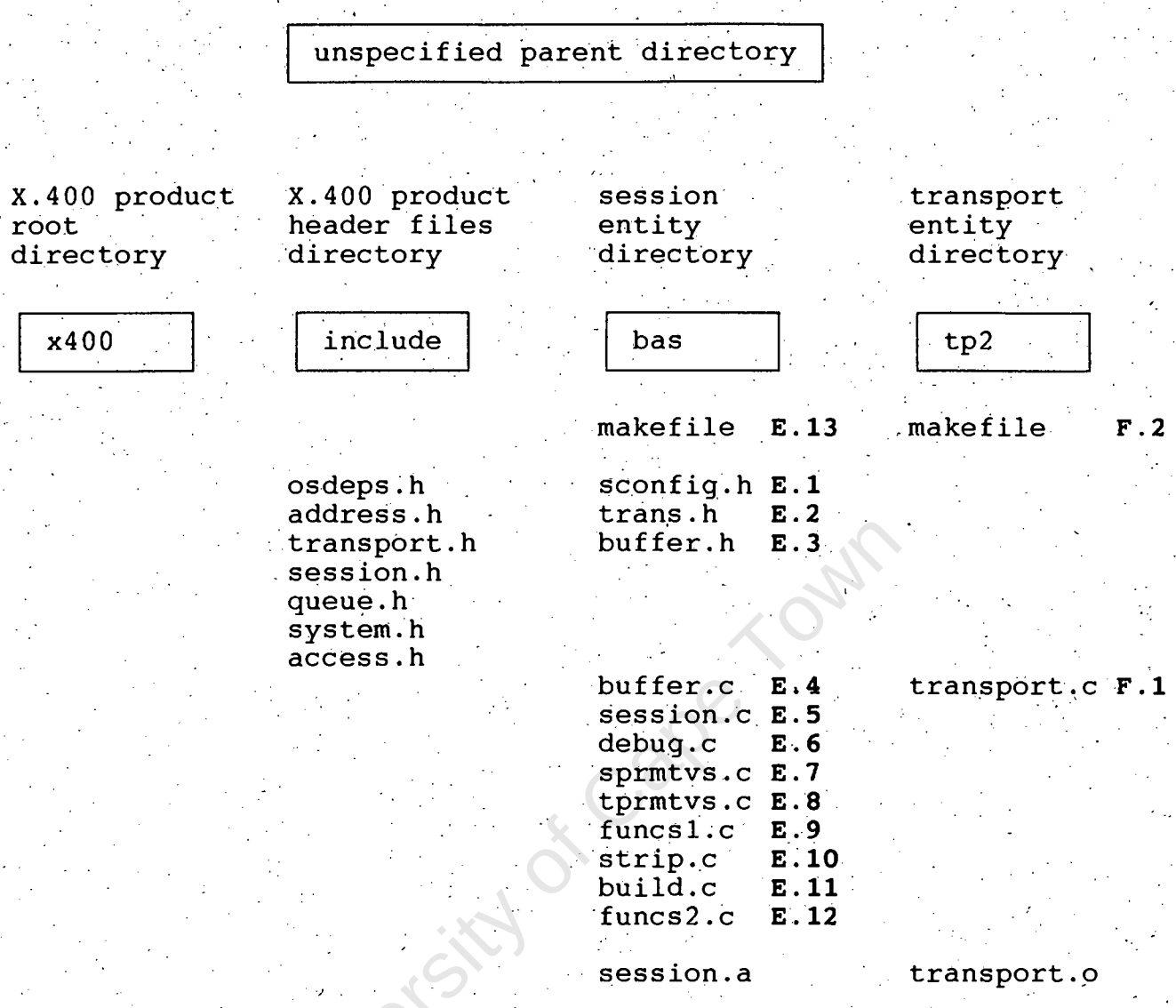

Figure 7.2 The file structure

Figure 7.2 does not show any of the X.400 product's source files, since they are beyond the scope of this project. Suffice to say, they all reside within the **x400** directory. The only X.400 product files indicated are seven header files in the **include** directory which are included by the session and transport entity source files.

The **bas** directory contains all files related to the session entity. The archive file **bas/session.a** contains two members: one compiled from **bas/buffer.c** and the other

compiled from **bas/session.c**. All other files with a **.c** extension are included by **bas/session.c**.

The **tp2** directory contains all files related to the transport entity. The object file **tp2/transport.o** is compiled from **tp2/transport.c**.

7.3.6 Programming conventions

As an aid to reading the source file listings, the following programming conventions should be noted:

a) Figure 7.3 illustrates the internal layout of the three main source files: **bas/buffer.c**, **bas/session.c** and **tp2/transport.c**.

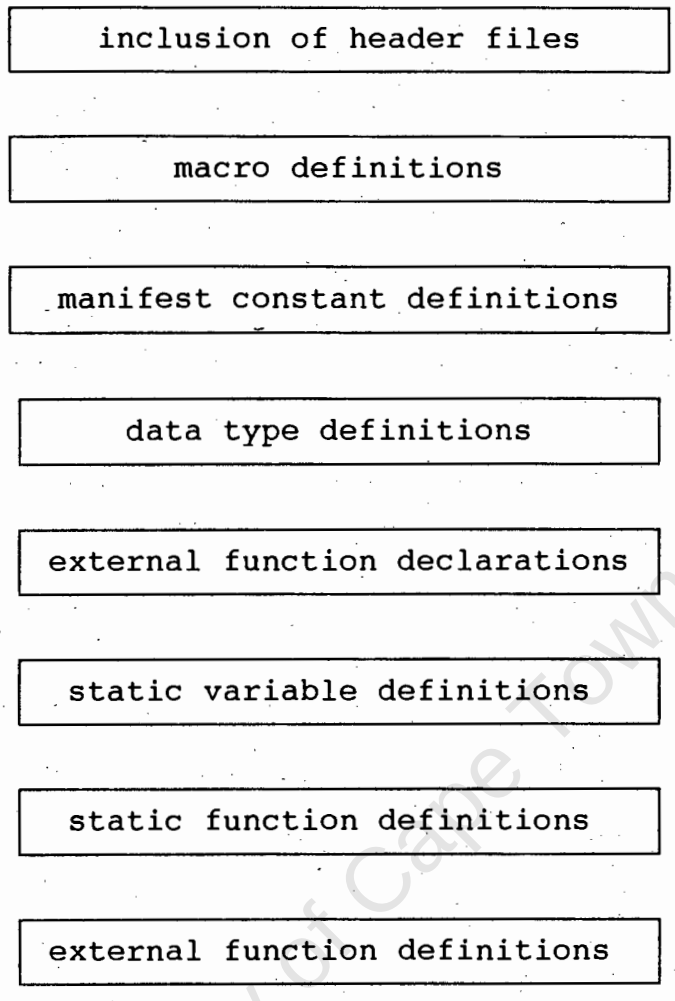

Figure 7.3 Internal layout of source files

b) All functions and variables which are only used locally within a source file are expressly assigned the **static** storage type. This avoids possible clashes with identical function and variable names in other RTS files. The only functions defined to be external (i.e., without a storage type identifier) are those which are called from other RTS source files.

There are no external variable definitions in any of these source files because the RTS, session entity and transport entity do not communicate via any external variables, only by means of function calls.

c) The syntax and semantics of functions, constants and variables used in the session entity source code are as similar as possible to those of the Estelle specification of section 6.

d) Those code sections which implement the session protocol state transitions are based closely on the transition declarations of the Estelle specification of section 6.

7.4 The X.400 product software facilities

The X.400 product provides several common facilities which are available to its software modules and to any other modules that interface with them. These facilities are provided in function libraries and header files.

The function libraries define external functions which are compiled into all X.400 product software modules. This makes them available to any other (user) modules which are compiled into the X.400 modules. In this manner, one copy of each facility is shared between all the modules of a compiled X.400 module. The header files may be included where needed.

Of all the common facilities available, this subsection briefly describes only those required by the session entity.

7.4.1 The interface to the operating system

To simplify porting, the X.400 software avoids direct calls to the operating system. Instead, it provides an environment-independent interface to the operating system through use of a library of environment-dependent functions and macros. These map pseudo operating system calls into actual operating system calls. Porting the X.400 software simply requires rewriting the functions and redefining the macros of this library.

a) The interface library functions:
The session entity requires only one of these functions:

```
void bcopy(from,to,length)
char *from,*to;
int  length;
```

This function copies **length** bytes from the buffer pointed to by **from** to the buffer pointed to by **to**. This function is provided in case the target operating system does not have a fast copy routine.

A significant portion of execution time is typically spent in this function. System performance may therefore be improved by optimizing this function. Although the X.400 product provides an environment-independent version of this function, it should be re-coded in the assembly language of the target operating system.

b) The interface library macros:
The header file **include/osdeps.h** defines all environment-dependent data types, defines macros which map certain pseudo-calls to their programming language library and system call equivalents, and includes all the associated standard header files. In addition, it defines a number of convenient manifest constants and environment-independent data types.

7.4.2 Common module interface definitions

To ensure standard, well-defined interfaces between the RTS and other modules which interface with it, three header files are provided which define various interface data types and manifest constants:

a) The header file **include/address.h:**
This file defines a uniform set of environment-independent data types for representing various OSI SAP address components.

Instead of adhering strictly to the one-to-one mapping between SSAP and TSAP addresses for X.400, the RTS software makes full use of the hierarchical addressing features of the OSI Reference Model. Therefore, this file defines data types for:

NSAP address,
TSAP selector,
SSAP selector,
TSAP address (a combination of the first two), and
SSAP address (a combination of the first three).

b) The header file **include/transport.h:**
This file provides environment-independent definitions for the interface between the session entity and the transport entity.

c) The header file **include/session.h:**
This file provides environment-independent definitions for the interface between the RTS and the session entity.

7.4.3 Queue management facilities

The X.400 software makes extensive use of doubly-linked circular lists. Therefore, a set of environment-independent functions for manipulating such data structures is provided. The header file **include/queue.h** contains external declarations for the queue management functions.

These functions operate on an *abstract* data type, **QuElement:**

```
typedef struct QuElement {
        struct QuElement *next;  /* next element */
        struct QuElement *prev;  /* previous element */
        /* any other user-defined members declared here */
} QuElement;
```

Objects that queue management functions manipulate are expected to have a **QuElement** casted over them, i.e., they must be structures with any number of members, as long as the first two members are pointers to the structure type. The queue management functions are:

```
QInit(p)
QuElement *q;

QInsert(q1,q2)
QuElement *q1,*q2;

QRemove(q)
QuElement *q;

QMove(q1,q2)
QuElement *q1,*q2;
```

QInit initializes a **QuElement** so that it can be used in subsequent operations. A **QuElement** is initialized by having its **next** and **prev** fields point to itself, thereby representing an empty circular list.

QInsert inserts the linked list **q1** before the linked list **q2**, resulting in a linked list that contains all members of **q1** and **q2**.

QRemove removes **QuElement q** from the list to which it belonged and initializes it upon completion of the removal.

QMove moves **QuElement q1** to the end of list **q2**.

7.4.4 SAP address comparison facilities

The X.400 software provides an environment-independent set of functions for the comparison of SAP selectors and NSAP addresses. No header file containing external declarations for these functions is provided. These functions are:

```
ssap_cmp(s1,s2)
ssap_selector *s1,*s2;

tsap_cmp(t1,t2)
tsap_selector *t1,*t2;

nsap_cmp(n1,n2)
nsap_address *n1,*n2;
```

These functions each expect two SAP selector/address data types and return zero if they were identical and non-zero if not.

7.4.5 Timer management facilities

The X.400 software provides an environment-independent interface to a set of timer facilities through the use of a set of environment-dependent functions. The header file **include/system.h** defines some environment-dependent manifest constants and macros which are used by these functions.

Six functions provide access to the timer facilities:

a) Timer module initialization:

init_memory()
init_timers()

Although not strictly part of the timer module, the function **init_memory** must be called only once, before any timer module function is called. This function reserves an area of memory for the various X.400 common facilities, including the timer module.

The function **init_timers** initializes the timer module. It must be called only once, before any other timer

module function is called. This function creates a circular list containing a fixed number (200) of available timers. This pool of available timers is shared between the RTS, session entity and transport entity.

b) Time progression:

clock()

The timers measure the progress of time in terms of 'clock ticks', where each 'clock tick' is generated by a separate call to this function. It is therefore the task of the timer module user to call this function periodically (the period is user-defined) to generate periodic 'clock ticks'. The intention is that the user should arrange for this function to be called automatically by some periodic, system interrupt.

c) Timer creation:

```
newtimer(machp,name,time,subscript,datum,func)
char *machp;
int name,time,subscript,datum;
int (*func)();
```

This function creates a new timer. The **time** argument specifies the timer's time period in terms of 'clock ticks'. This function arranges for the user-supplied function **func** to be called when the timer expires. When this happens, attributes of the timer are passed to **func** as follows:

```
(*func)(machp,name,subscript,datum);
```

period, it will be interrupted itself and chaos will result.

The header file **include/system.h** defines two items of interest to the timer module:

a) The **CLOCK** constant:
This constant is defined to be the (user-defined) period of the 'clock-ticks' in milliseconds. It is used to scale absolute milliseconds to the corresponding number of 'clock ticks', thereby providing the user with a convenient way of loading a timer in terms of milliseconds. Naturally, the user cannot specify a time period in milliseconds which is beyond the resolution determined by the period of the 'clock ticks'.

For example, to set a timer for 5 seconds, the **time** argument of the **newtimer** function becomes: **5000/CLOCK**.

b) The macros: **ENTER()** and **LEAVE()**:
The timer module functions **clock**, **newtimer**, **cantimer** and **do_timer_queue** all access the queue of active timers. However, **clock** is invoked at interrupt time while the other three are not. This may result in variable access contention between **clock** and the other three functions.

To avoid this problem, the two macros **ENTER()** and **LEAVE()** are provided. They are used to protect the critical sections of **newtimer**, **cantimer** and **do_timer_queue** against pre-emption (interrupt) by **clock**. They must therefore behave as Interrupt Disable and Interrupt Enable instructions, respectively. Because the interrupt mechanism for **clock** is an environment-dependent, user-defined issue, the definition of these two macros is too. They are

therefore provided as undefined macros for definition by the user.

7.4.6 PDU parsing and formatting facilities

The X.400 software provides an environment-independent set of PDU parsing and formatting facilities through the use of a set of environment-dependent macros. The header file **include/access.h** contains these macros. These macros are:

add1(pdu,value)
add2(pdu,value)
add4(pdu,value)
adds(pdu,data,length)

get1(value,pdu)
get2(value,pdu)
get4(value,pdu)
gets(pdu,data,length)

add1 appends the 8 bit integer **value** to the PDU, at the byte pointed to by **pdu**, and post-increments **pdu** by 1 byte.

add2 appends the 16 bit integer **value** to the PDU, starting at the byte pointed to by **pdu**, in the order of Most Significant Byte (MSB) then Least Significant Byte (LSB), and post-increments **pdu** by 2 bytes.

add4 appends the 32 bit integer **value** to the PDU, starting at the byte pointed to by **pdu**, in the order of MSB to LSB, and post-increments **pdu** by 4 bytes.

adds appends **length** bytes, starting at the byte pointed to by **data**, to the PDU, starting at the byte pointed to by **pdu**, and post-increments **pdu** by **length** bytes.

get1 parses the contents of the PDU, at the byte pointed to by **pdu**, into an 8 bit integer **value**, and post-increments **pdu** by 1 byte.

get2 parses the contents of the PDU, starting at the byte pointed to by **pdu**, into a 16 bit integer **value**, assuming that PDU is in the order of MSB then LSB, and post-increments **pdu** by 2 bytes.

get4 parses the contents of the PDU, starting at the byte pointed to by **pdu**, into a 32 bit integer **value**, assuming that PDU is in the order of MSB to LSB, and post-increments **pdu** by 4 bytes.

gets copies **length** bytes from the PDU, starting at the byte pointed to by **pdu**, to the buffer starting at the byte pointed to by **data**, and post increments **pdu** by **length**-bytes.

These macros must be adopted to the target environment. The file **include/access.h** contains a set of conditionally compiled macros for the more popular processors. The appropriate set must be selected by defining the appropriate manifest constant in this file.

7.4.7 Exception handling facilities

The X.400 product provides an environment-dependent function for handling unrecoverable exception conditions:

void fatal()

This function should be called when an unrecoverable problem, such as the lack of an essential resource, is detected. If the environment provides facilities for obtaining core dumps on resets, these facilities should be used to take appropriate action. The X.400 software does

not define **fatal**, as this is an environment-dependent issue. This facility must therefore be defined by the user.

The function **fatal** is called twice by the timer module functions **init_timers** and **newtimer**. The function **init_timers** calls **fatal** if not enough memory can be allocated for the timer module, while **newtimer** calls **fatal** if the timer module has run out of available timers.

Since **fatal** is only called from the timer module, it was decided to define it as a macro in the timer module header file, **include/system.h**. This macro simply prints an error message and exits. The necessary additions to **include/access.h** are as follows:

```
#include "osdeps.h"  /* for os_exit */
#define fatal() {printf("Fatal timer error, so exit.\n");
                 os_exit(0);}
```

7.5 The session layer interface

This subsection defines the interface between the RTS and the session entity. This interface consists of ten external functions which reside in the RTS and session entity modules. Calls to all of these functions are asynchronous and non-blocking. Figure 7.4 illustrates the logical positioning of these functions in relation to one another.

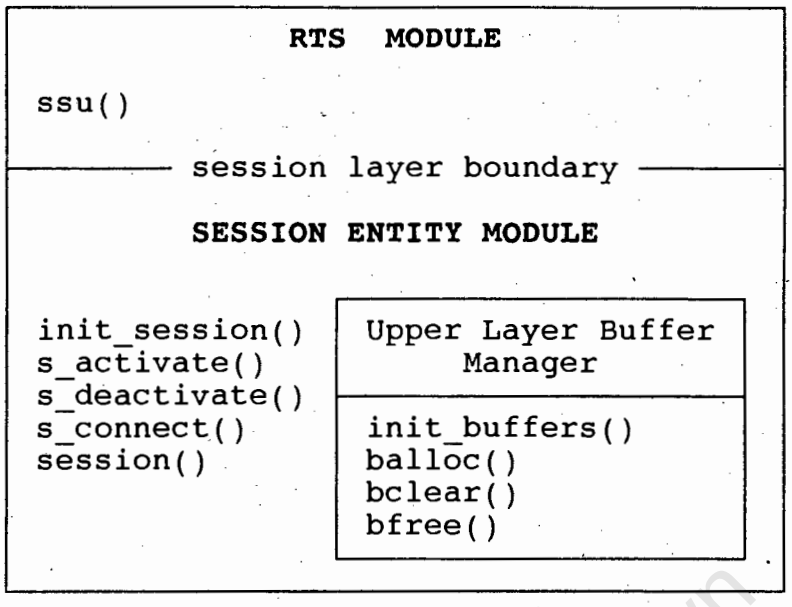

Figure 7.4 The session layer interface functions

7.5.1 The upper layer buffer manager

Before describing the buffer manager, it is important to note that buffer management is strictly a local matter and therefore not subject to any OSI specification.

It is normally desirable for the upper layers (session and above) to use common buffer management routines and a common buffer pool. The objectives of these routines is quite different from those of the transport layer and below. The transport layer needs many small buffers, each containing one TPDU (typically around 1024 bytes long), while the session and higher layers use a few, relatively large buffers whose maximum size is determined by the application.

The RTS module expects the session entity module to provide an upper layer buffer manager. This buffer manager creates a static pool of buffers and provides its users with the means to allocate and de-allocate the buffers

Each buffer is accessed via its *buffer descriptor*, a static structure maintained in memory which contains all information concerning the buffer. The buffer descriptor data type, **struct buf**, is defined in **include/session.h**.

These buffers are intended to be used by the application for user data. In order to minimize processing time, it is important that applications build data buffers from the end, leaving space in front of the data, so that the OSI protocol layers can prepend protocol information headers without having to move the data. The buffer manager supports this concept by having each buffer descriptor store two pointers to the buffer: a fixed pointer to the beginning of the buffer and a variable pointer to the first used byte of the buffer.

The external functions by which the buffer manager is accessed are the following:

a) Buffer pool initialization:

int init_buffers(nbufs,bufsz)
int nbufs,bufsz;

This function initializes the buffer pool by reserving memory for it and setting up **nbufs** buffers, each of size **bufsz** bytes. This function must be called only once by the RTS, before it makes any other calls to the session entity. If unsuccessful, i.e., if not enough memory is available for the buffer pool, this function returns zero. Otherwise, it returns a non-zero value.

It is recommended that the number of buffers created should be at least twice the number of simultaneous session connections that the session entity can support, plus a few extra. This allows one buffer for an incoming SSDU and one for an outgoing SSDU per

session connection. In fact, the RTS module assumes the availability of 16 simultaneous session connections and therefore requests the creation of 36 (16 * 2 + 4) buffers.

Each buffer must be large enough to hold a complete TSDU, except in the case of unlimited-length TSDUs. In fact, the RTS module requests that the buffer size be:

BSIZE = CHECKSIZE * 1024 + 10

bytes, where **CHECKSIZE** is the RTS checkpoint-size parameter (see 4.11.1) which specifies the maximum SSDU size in kilobytes. The extra 10 bytes are for holding a DATA TRANSFER SPDU header (with a maximum length of 7 bytes) and a concatenated GIVE TOKENS SPDU (with a length of 3 bytes) in one TSDU. This buffer size therefore represents the maximum TSDU size that the session entity can handle. This is the value that the session entity will propose in the Maximum TSDU Size parameters of the CONNECT and ACCEPT SPDUs.

If the RTS checkpoint-size parameter is zero, it indicates that the RTS uses unlimited-length SSDUs. In this case, the session entity must provide local segmenting of unlimited-length SSDUs across its upper interface. Note that if the RTS checkpoint-size parameter is zero, it does not necessarily mean that the maximum buffer size becomes 10! In this case, the maximum buffer size should be assigned an appropriate default value to contain local SSDU segments. This default value must be large enough to hold any complete TSDU not containing a DATA TRANSFER SPDU.

Similarly, if the selected maximum TSDU size for the transport connection, as negotiated between the correspondent SPMs, is zero (i.e., unlimited-length), the session entity will have to provide local

segmenting of unlimited-length TSDUs across its lower interface.

These two local SSDU and TSDU segmenting issues must not be confused with the end-to-end segmenting of DATA TRANSFER SPDUs which are too large to fit into one TSDU. The former are strictly local matters while the latter forms part of the session protocol.

b) Buffer allocation:

struct buf *balloc(n)
int n;

This function allocates a buffer from the pool and initializes its **length** and **addr** fields to create a buffer of length **n**. The **eosdu** field is initialized to **TRUE**. If the requested length is negative, this function will initialize the buffer to the maximum length possible (**bufsz**). This feature is used by the session entity when it needs to allocate a new receive buffer and does not know the length of the next incoming SSDU.

If successful, this function returns a pointer to the allocated buffer's buffer descriptor. Otherwise, i.e., if the buffer pool is empty, it returns the **NULL** pointer.

c) Buffer clear:

void bclear(bp)
struct buf *bp;

This function resets a buffer descriptor's **addr** and **length** fields to indicate a buffer of zero length.

d) Buffer de-allocation:

void bfree(bp)
struct buf *bp;

This function returns a previously allocated buffer to the pool.

The source code for the buffer manager is contained in **bas/buffer.c**, while the header file **bas/buffer.h** contains external declarations for the buffer manager.

7.5.2 Session entity initialization

Before the session entity can accept the registration of an SS-user or provide any session services, it must first be initialized. This action is strictly a local matter and is therefore not subject to any OSI specification. It is accomplished by an SS-user call to the following session entity function:

void init_session(tsap_id)
tsap_selector *tsap_id;

This function intializes the session entity's queues, timers, transport layer interface, etc. It must be the first session entity function called by the SS-user and must be called only once.

The SS-user specifies the local TSAP selector for its supporting session entity through the pointer **tsap_id**. The session entity uses this TSAP selector when registering itself with its supporting transport entity (see 7.6.1). This function is defined in **bas/session.c**.

7.5.3 SS-user registration

Before an SS-user can use any session services, it must first register itself with its supporting session entity. This action is analogous to an SS-user attaching itself to an SSAP in the OSI Reference Model. To register, an SS-user calls the following session entity function:

```
int s_activate(ssap_id,ssu)
ssap_selector *ssap_id;
void (*ssu)();
```

The SS-user specifies its local SSAP selector through the pointer **ssap_id**. If the specified SSAP selector has zero length, the session entity assumes a one-to-one session address mapping for the SS-user.

Although X.400 specifies a one-to-one mapping between SSAP and TSAP addresses, the RTS module, because it supports full OSI hierarchical address mapping, may specify a SSAP selector when it registers as an SS-user. It is not necessarily wrong for it to do so, the session entity simply ignores this value. The fact that the session entity is supplied with only one TSAP selector at initialization time, coupled with the fact that X.400 specifies a one-to-one mapping between session and transport addresses, restricts the session entity to allowing only one RTS to register with it.

The pointer **ssu** points to the SS-user function which the session entity must call in future in order to pass session indication and confirm primitives up to it (see 7.5.7).

If this function completes successfully, i.e., if only one RTS has registered, it returns a non-zero value. Otherwise, it returns zero. This function is defined in **bas/session.c**.

7.5.4 SS-user de-registration

When the SS-user has finished using session services, it de-registers itself from its supporting session entity. This action is analogous to an SS-user detaching itself from an SSAP in the OSI Reference Model. To de-register, an SS-user calls the following session entity function:

```
int s_deactivate(ssap_id)
ssap_selector *ssap_id;
```

The SS-user specifies its SSAP selector through the pointer **ssap_id**. As mentioned before, SSAP selectors specified by the RTS are ignored.

After this call, the SS-user cannot initiate or participate in session connections. If it has any active session connections at the time of this call, the call will fail and return zero. Otherwise, it returns a non-zero value.

This function is defined in **bas/session.c**. A flowchart showing the internal operation of **s_deactivate** is presented in Figure 7.9 in subsection 7.8.4. The reason for this placing will become clear once subsection 7.8.4 has been read.

7.5.5 Session connection initiation

The SS-user calls the following session entity function to pass a S-CONNECT.request primitive down to it:

```
struct Smachine *s_connect(idu)
struct idu *idu;
```

This function attempts to allocate a SPM for the session connection and, if successful, initiates session connection establishment.

The structure **struct idu** represents a Session Interface Data Unit (SIDU) and is used by the session layer interface to carry all data associated with any session service primitive. It has one member which identifies the primitive concerned, and several other members which hold the parameter values of all session service primitives. The type definition for this structure, together with the type definitions of its members, is provided in the header file **include/session.h**. This structure is defined as a local variable in the calling function. Therefore, any variables contained in it that must be saved by the session entity should be copied away from it before returning to the calling function.

In this implementation, an SPM is a static structure maintained in memory by the session entity on a per connection basis. It contains all state information for a particular connection. This structure is of type **struct Smachine** and is defined only within the session entity. A pointer to such a structure represents the SCEP identifier of the session connection with which it is associated.

If successful, this function returns a pointer (the SCEP identifier) to the allocated SPM structure. The SS-user will in future use this value only as a SCEP identifier and never as a pointer. If unsuccessful, i.e., if the session connection cannot be initiated due to lack of resources or some other local session error, this function returns the **NULL** pointer.

Unlike the session entity described by the Estelle specification of section 6, the one implemented here performs a dynamic allocation of SPMs to session and transport connections, as required.

This function is defined in **bas/session.c**. A flowchart showing the internal operation of **s_connect** is presented in Figure 7.6 in subsection 7.8.4. The reason for this placing will become clear once subsection 7.8.4 has been read.

7.5.6 Session request and response primitives

The SS-user calls a single session entity function to pass all session request and response primitives (except S-CONNECT.request) down to it. It calls this function once for every session request and response primitive to be passed down to the session entity:

```
int session(s,idu)
struct Smachine *s;
struct idu      *idu;
```

The parameter **s** points to the SPM supporting this session connection and is used by the SS-user as the SCEP identifier. The implementation of SCEPs is not subject to any OSI specification and is therefore strictly a local matter.

The parameter **idu** points to the SIDU. Its **event** member identifies the particular session service primitive being invoked, while its other members hold the relevant parameter values associated with the particular primitive.

This function is responsible for performing all the required processing, primitive calls and state transitions required by all incoming session request and response primitives. If this function is unsuccessful i.e., if some local error prevents it from completing the primitive invocation, it returns zero. Otherwise, it returns a non-zero value.

This method of implementing session primitive invocations has one major strength: only one function and one data structure are used to invoke all of them, excluding S-CONNECT.request. Because there are so many of these primitives, many with long parameter lists, it would be very inefficient from a coding point of view if the session entity were to provide separate functions to handle these different primitives. In addition, this method provides a neat, efficient way of implementing SCEP identifiers.

This function is defined in **bas/session.c**. A flowchart showing the internal operation of **session** is presented in Figure 7.7 in subsection 7.8.4. The reason for this placing will become clear once subsection 7.8.4 has been read.

7.5.7 Session indication and confirm primitives

The session entity calls a single SS-user function to pass all session indication and confirm primitives up to it. The SS-user provided the session entity with a pointer to this function in its (earlier) call to **s_activate**. The session entity calls this function once for every session indication and confirm primitive to be passed up to the SS-user. If this function is named **ssu** it is called as follows:

```
void (*ssu)(s,idu)
struct Smachine *s;
struct idu      *idu;
```

As with the session entity function **session**, the parameter **s** represents the SCEP identifier, while **idu** points to a SIDU structure. This structure is defined as a local variable in the calling function. Therefore, any variables contained in it that must be saved by the SS-user should

be copied away from it before returning to the calling function.

The session entity does not call this function *directly* wherever needed. Instead, it calls it via a set of internal, static functions, one for each session indication or confirm primitive. There are two reasons for doing this: Firstly, certain primitives always have the same values for certain parameters, and secondly, it is desirable to have debug statements printed for every invocation of a particular primitive during testing. With this scheme, only a single copy of both the 'constant' parameter values and the debug code need exist for a particular primitive (within the primitive calling function) instead of at every place in the program from where the primitive is invoked. This set of primitive calling functions is defined in the source file **bas/sprmtvs.c**.

7.6 The transport layer interface

This subsection defines the interface between the session entity and the transport entity. Unlike the session layer interface, this interface is not intended to be compatible with any existing transport layer product. Rather, it shows an example implementation.

This interface consists of nine external functions which reside in the session and transport entities. Calls to all of these functions are asynchronous and non-blocking. Figure 7.5 illustrates the logical positioning of these functions in relation to one another.

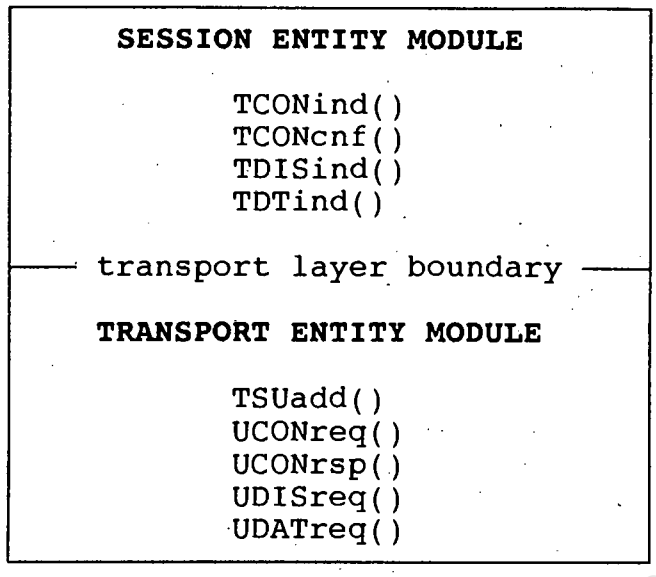

Figure 7.5 The transport layer interface functions

All the session entity functions are defined in **bas/session.c**, while all the transport entity functions are defined in **tp2/transport.c**.

7.6.1 TS-user registration

Before a TS-user can use any transport services, it must first register itself with its supporting transport entity. This action is analogous to a TS-user attaching itself to a TSAP in the OSI Reference Model. To register, a TS-user calls the following transport entity function:

```
void TSUadd(tsap_id,tconind,tconcnf,tdisind,tdtind)
tsap_selector *tsap_id;
int (*tconind)();   /* T-CONNECT.indication handler */
int (*tconcnf)();   /* T-CONNECT.confirm handler */
int (*tdisind)();   /* T-DISCONNECT.indication handler */
int (*tdtind)();    /* T-DATA.indication handler */
```

The TS-user specifies its local TSAP selector through the **tsap_id** pointer. The other four function parameters are all pointers to TS-user functions which the transport entity must call in future in order to pass transport indication and confirm primitives up to the session entity.

The session entity calls this function during session entity initialization, i.e., during an SS-user' call to **init_session**. The SS-user supplies the session entity with its local TSAP selector in this call, which the session entity then provides to its supporting transport entity in its call to **TSUadd**. The session entity supplies the addresses of its **TCONind**, **TCONcnf**, **TDISind** and **TDTind** functions, respectively, as the pointers to the TS-user transport indication and confirm primitive handlers.

Unlike the session layer interface, the TS-user provides a separate handler function for each different incoming transport primitive. The reason for this is that, unlike the session service primitives, there are only a few transport service primitives, each with short parameter lists, so that providing separate handler functions for each is quite feasible in terms of coding complexity.

7.6.2 The T-CONNECT.request primitive

The TS-user calls the following transport entity function to pass the T-CONNECT.request primitive down to it:

```
pointer UCONreq(clgTSAPid,
                cldNSAPaddr,
                cldTSAPid,
                qots,
                texp,
                TSUdata)

tsap_selector *clgTSAPid;     /* calling TSAP selector */
nsap_address  *cldNSAPaddr;   /* called  NSAP address */
tsap_selector *cldTSAPid;     /* called  TSAP selector */
qos_type      *qots;          /* proposed QOTS */
boolean       texp;           /* proposed TEXP option */
uint8         *TSUdata;       /* TS-user data */
```

The TS-user provides only the calling TSAP selector element of the calling TSAP address because only the transport entity knows what the NSAP address component is. On the other hand, the TS-user supplies the full called TSAP address by means of its two components: called NSAP address and called TSAP selector.

In this implementation, no use is made of the QOTS or TS-user Data parameters of transport service primitives. Therefore, the session entity will always assign the **NULL** pointer to the **qos** and **TSUdata** parameters.

If successful, this function returns a pointer which the session entity uses as the TCEP identifier. If unsuccessful, i.e., if the transport connection cannot be initiated due to a lack of resources or some other local transport error, this function returns the **NULL** pointer.

7.6.3 The T-CONNECT.response primitive

The TS-user calls the following transport entity function to pass the T-CONNECT.response primitive down to it:

```
void UCONres(TCEPid,qots,texp,TSUdata)
pointer  TCEPid;     /* TCEP identifier */
qos_type *qots;      /* selected QOTS */
boolean  texp;       /* selected TEXP option */
uint8    *TSUdata;   /* TS-user data */
```

In this implementation, no use is made of the QOTS or TS-user Data parameters of transport service primitives. Therefore, the session entity will always assign the **NULL** pointer to the **qos** and **TSUdata** parameters.

7.6.4 The T-DISCONNECT.request primitive

The session entity calls the following transport entity function to pass the T-DISCONNECT.request primitive down to it:

```
void UDISreq(TCEPid,TSUdata)
pointer TCEPid;     /* TCEP identifier */
uint8   *TSUdata;   /* TS-user data */
```

In this implementation, no use is made of the TS-user Data parameter of transport service primitives. Therefore, the session entity will always assign the **NULL** pointer to the **TSUdata** parameter.

7.6.5 The T-DATA.request primitive

The TS-user calls the following transport entity function to pass the T-DATA.request primitive down to it:

```
void UDATreq(TCEPid,tsdu,eotsdu)
pointer    TCEPid;  /* TCEP identifier */
struct buf *tsdu;   /* TSDU buffer */
boolean    eotsdu;  /* end of TSDU flag */
```

The TSDU buffer contains a complete TSDU, except in the case where no segmenting has been selected for this end-to-end direction of transfer i.e., unlimited-length TSDUs are used. In this case, the TS-user may locally segment a long TSDU (i.e., one which does not fit into one TSDU buffer) and transfer it across this interface in several calls to this function. The **eotsdu** flag is set to **TRUE** if the TSDU buffer contains a complete TSDU or the last segment of a locally segmented TSDU. Otherwise, it is set to **FALSE**. This local segmentation of TSDUs is strictly a local matter and is therefore not subject to any OSI specification.
This local segmentation must not be confused with the end-to-end segmentation of unlimited-length DATA TRANSFER SPDUs. This only occurs when segmenting has been selected for this end-to-end direction of transfer and TSDUs are of finite size. It may thus be seen that the local segmenting of TSDUs and the end-to-end segmenting of DATA TRANSFER SPDUs are mutually exclusive.

7.6.6 The T-CONNECT.indication primitive

The transport entity calls the following TS-user function (by means of a pointer supplied by the TS-user in a call to **TSUadd**) to pass the T-CONNECT.indication primitive up to it:

```
int TCONind(TCEPid,
            clgNSAPaddr,
            clgTSAPid,
            cldTSAPid,
            qots,
            texp,
            TSUdata)

pointer       TCEPid;        /* TCEP identifier */
nsap_address  *clgNSAPaddr;  /* calling NSAP address */
tsap_selector *clgTSAPid;    /* calling TSAP selector */
tsap_selector *cldTSAPid;    /* called  TSAP selector */
qos_type      *qots;         /* proposed QOTS */
boolean       texp;          /* proposed TEXP option */
uint8         *TSUdata;      /* TS-user data */
```

This function always returns the value **TRUE**.

Of the called TSAP address, the transport entity only indicates the called TSAP selector component because only it needs to know what the called NSAP address component is. On the other hand, the transport entity supplies the full calling TSAP address by means of its two components: calling NSAP address and calling TSAP selector.

In this implementation, no use is made of the QOTS or TS-user Data parameters of transport service primitives. Therefore, the transport entity will always assign the **NULL** pointer to the **qos** and **TSUdata** parameters.

7.6.7 The T-CONNECT.confirm primitive

The transport entity calls the following TS-user function (by means of a pointer supplied by the TS-user in a call to **TSUadd**) to pass the T-CONNECT.confirm primitive up to it:

```
int TCONcnf(TCEPid,
            cldNSAPaddr,
            cldTSAPid,
            qots,
            texp,
            TSUdata)

pointer       TCEPid;        /* TCEP identifier */
nsap_address  *cldNSAPaddr;  /* called NSAP address */
tsap_selector *cldTSAPid;    /* called TSAP selector */
qos_type      *qots;         /* proposed QOTS */
boolean       texp;          /* proposed TEXP option */
uint8         *TSUdata;      /* TS-user data */
```

This function always returns the **TRUE** value.

The transport entity supplies the full called TSAP address by means of its two components: called NSAP address and called TSAP selector.

In this implementation, no use is made of the QOTS or TS-user Data parameters of transport service primitives. Therefore, the transport entity will always assign the **NULL** pointer to the **qos** and **TSUdata** parameters.

7.6.8 The T-DISCONNECT.indication primitive

The transport entity calls the following TS-user function (by means of a pointer supplied by the TS-user in a call to **TSUadd**) to pass the T-DISCONNECT.indication primitive up to it:

```
int TDISind(TCEPid,reason,TSUdata)
pointer TCEPid;     /* TCEP identifier */
uint8   reason;     /* disconnect reason */
uint8   *TSUdata;   /* TS-user data */
```

This function always returns the **TRUE** value.

In this implementation, no use is made of the TS-user Data parameter of transport service primitives. Therefore, the session entity will always assign the **NULL** pointer to the **TSUdata** parameter.

7.6.9 The T-DATA.indication primitive

The transport entity calls the following TS-user function (by means of a pointer supplied by the TS-user in a call to **TSUadd**) to pass the T-DATA.indication primitive up to it:

```
int TDTind(TCEPid,tsdu,eotsdu)
pointer    TCEPid;  /* TCEP identifier */
struct buf *tsdu;   /* TSDU buffer */
boolean    eotsdu;  /* end of TSDU buffer flag */
```

This function always returns the **TRUE** value.

The use of the TSDU buffer is identical to that of the TSDU buffer in the T-DATA.request primitive function call, except that the transfer direction is now from TS-provider to TS-user.

7.7 Receiving data from the transport layer

One issue to be resolved in this implementation is to determine *when* the lower layer entities call the upper layer entities. This may be either only when explicitly requested to do so or simply whenever data arrives. The latter option is assumed, although not explicitly specified, by the OSI specifications. Any decision taken in this regard is therefore strictly a local matter.

The approach adopted in this implementation is that lower layer entities only call upper layer entities when explicitly requested to do so by upper layer entities. This approach implies a flow-control scheme which allows the upper layer entities to prevent themselves from being overloaded with data.

This scheme is implemented by the session entity in two phases:

a) Incoming transport indication and confirm primitives are stored in a FIFO queue of incoming transport events. In practical terms, this means that all the parameters associated with a transport entity call to one of the session entity's **TCONind**, **TCONcnf**, **TDISind** or **TDTind** functions are stored, together with a primitive identifier, as one element on a doubly-linked circular list.

b) The session entity does not process the primitives in the transport event queue until explicitly instructed to do so by the SS-user. The following external session entity function is provided for this purpose:

   void do_session_queue()

   When the SS-user calls this function, the session entity sequentially removes each transport event from the queue and performs all the required processing, primitive calls and state transitions for that event. This function only returns once all the events on the queue have been processed and the queue is empty. The SS-user may then check its own queue of inbound data to see if anything was actually received.

This scheme represents a continuous polling by the SS-user of its input channels. Because **do_session_queue** returns immediately if (or once) the transport event queue is empty, it allows the SS-user to perform other activities while waiting for incoming data.

This function is defined in **bas/session.c**. A flowchart showing its internal operation is presented in Figure 7.8 in subsection 7.8.4. The reason for this placing will become clear once subsection 7.8.4 has been read.

7.8 SPM timers

In this implementation, each SPM uses two timers:

a) The standard session protocol timer, TIM. This timer will be termed the *abort timer*.

b) A timer which controls the release of a reusable transport connection. This timer will be termed the *connect timer*.

The connect timer is not required or specified by the Session Protocol Specification of CCITT Recommendation X.225, therefore its use and implementation is strictly a local matter.

This timer is started whenever the SPM enters state STA01C (idle, reusable transport connection). If this transport connection is reused by a new session connection before this timer expires, this timer is cancelled. If this timer expires, the SPM releases its reusable transport connection and enters state STA01 (the idle state).

This reason for using this timer is to avoid situations where reusable transport connections are maintained indefinitely by an SPM while the SS-user has no intention of re-establishing a session connection. Such situations represent a waste of valuable system resources.

Reusable transport connections are intended to be used by the session entity in situations where a SS-user releases a session connection (possibly due to some local or protocol error) with the intention of re-establishing it soon to the same remote session address. For this reason, the time period with which

the connect timer is loaded should be slightly greater that of the SS-user's typical session connection re-establishment delay. The exact delay value is clearly an implementation-dependent issue.

7.8.1 Timer definitions

The X.400 product header file **include/system.h** defines two manifest constants which define the time periods with which the two SPM timers are loaded:

a) **#define SLOWTIMER 60000/CLOCK** defines a 60 second period for the connect timer, while
b) **#define FASTTIMER 10000/CLOCK** defines a 10 second period for the abort timer.

These values are intended to be passed to the **newtimer** function call as the **time** parameter.

The session entity source file **bas/session.c** defines two manifest constants for identifying the two timer types:

a) **CONNECT_TIMER** identifies the connect timer, while
b) **ABORT_TIMER** identifies the abort timer.

These values are intended to be passed to the **newtimer** and **cantimer** function calls as the **name** parameter.

7.8.2 Stopping and starting the timers

The abort timer is stopped and started by the specific actions [3] and [4] respectively, as specified by CCITT Recommendation X.225. Because the connect timer is not part of this Recommendation, it is stopped and started by two local "specific actions" which are not defined in Recommendation X.225, namely specific actions [32] and [33], respectively. The source code for these specific actions is contained in **bas/funcs1.c**.

7.8.3 Processing expired timers

The following internal session entity function is called whenever an SPM timer expires:

```
static void SessionTimeOut(s,name,subscript,datum)
struct Smachine *s;
int             name,subscript,datum;
```

A pointer to this function is passed to the **newtimer** function as the **func** parameter whenever an SPM timer is created.

The **s** parameter points to the SPM which started the timer, **name** is the timer identifier as described earlier, while **subscript** and **datum** are not used by this implementation.

This function is responsible for performing all the required processing, primitive calls and state transitions required by expired timers. It is defined in the session entity source file **bas/session.c**.

7.8.4 The timer interrupt structure

Four issues must be resolved before using the X.400 product timer module:

a) The constant **CLOCK** must be defined.
b) The functions **init_memory** and **init_timers** must be called once.
c) The functions **clock** and **do_timer_queue** must be called periodically.
d) The macros **ENTER()** and **LEAVE()** must be defined.

The constant **CLOCK** has been defined as 1000 milliseconds, which becomes the period of the 'clock ticks'.

The RTS expects the session entity to call the functions **init_memory**, **init_timers**, **clock** and **do_timer_queue**.

At session entity initialization time, i.e., during an SS-user call to **init_session**, the session entity calls **init_memory** and **init_timers**. It then initiates the periodic calling of **clock** and **do_timer_queue** by calling the following internal function once:

static void ClockInterrupt()

This function is entirely responsible for periodically calling the functions **clock** and **do_timer_queue**, thus relieving the session entity from any further concern of these issues. This function arranges for itself to be called whenever the UNIX **signal** facility captures the **SIGALRM** interrupt. To generate this interrupt periodically, this function uses the UNIX **alarm** facility, which it resets each time it is called. The time period with which this function sets the UNIX **alarm** facility is one second, which is the period of one 'clock tick'. This function is defined in **bas/funcs1.c**.

It must be stressed that the RTS was checked for not using the **SIGALRM** interrupt before this scheme was implemented.

Note that **do_timer_queue** is called at interrupt time while, strictly speaking, this should not be so. This will, however, not cause any interrupt problems as long as **SessionTimeOut**, which is called by **do_timer_queue** upon detecting an expired timer, does not take longer than the interrupt period to execute. For this reason, the function **SessionTimeOut** is very short and simple and takes well under one second to execute.

As described in subsection 7.4.5, critical sections of the X.400 product's timer module functions must be protected against pre-emption by the interrupt that calls the **clock** function. In addition, the session entity itself also has critical sections that must be protected against this interrupt. These stem from the fact that the session entity functions **SessionTimeOut**, **s_deactivate**, **s_connect**, **session** and **do_session_queue** all access the SPM's **STATE** variable. **SessionTimeOut** is called at interrupt time only (by **do_timer_queue** if an expired timer is detected) while the other four functions are not. Clearly, this may lead to access contention for the SPM's **STATE** variable.

Note that the four functions **s_deactivate**, **s_connect**, **session** and **do_session_queue** do not threaten each other's critical sections. Because they are all called by the same process (RTS), they cannot be called simultaneously, making them inherently mutually exclusive.

All these critical sections are threatened by the same interrupt, **SIGALRM**. The timer module's critical sections are protected by the macros **ENTER()** and **LEAVE()**. Because these macros are defined in the X.400 product header file **include/system.h**, they may also be used to protect the session entity's critical sections. These macros have been

defined to act as Interrupt Disable and Interrupt Enable instructions, respectively, for the **SIGALRM** interrupt.

An important side-effect resulting from the protection of the session entity's critical sections is that the protocol state transitions contained in them become atomic (indivisible), as required by the FDT Estelle.

With this interrupt scheme explained, flowcharts may now be presented showing the internal operation of the session entity functions **s_connect, session, do_session_queue** and **s_deactivate. These flow charts are presented in Figures** 7.6 to 7.9 and emphasize the protection of critical code sections against the **SIGALRM** interrupt.

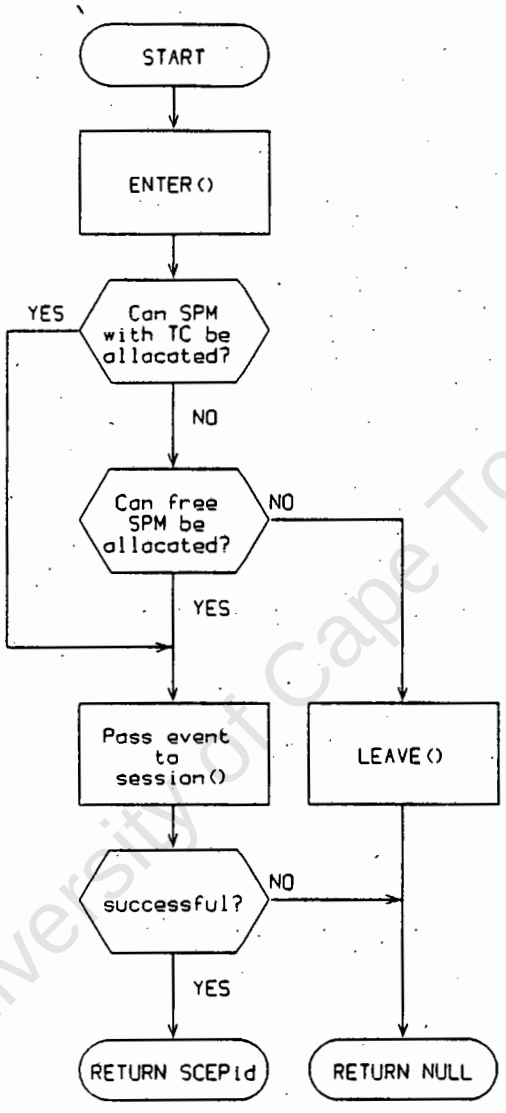

Figure 7.6 Flowchart for s_connect()

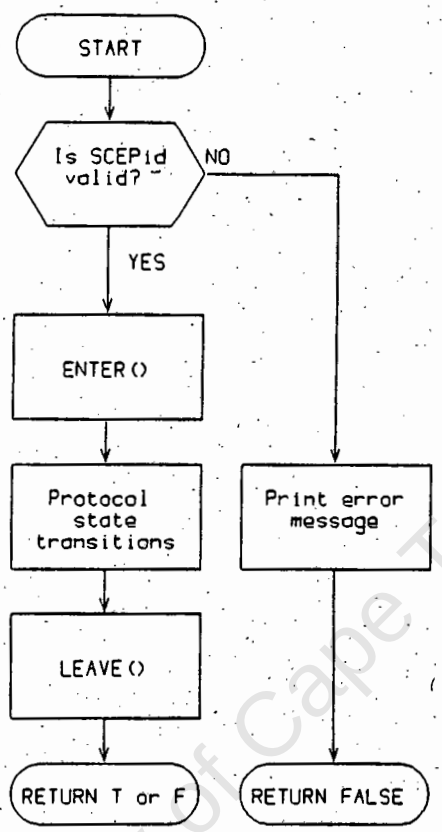

Figure 7.7 Flowchart for session()

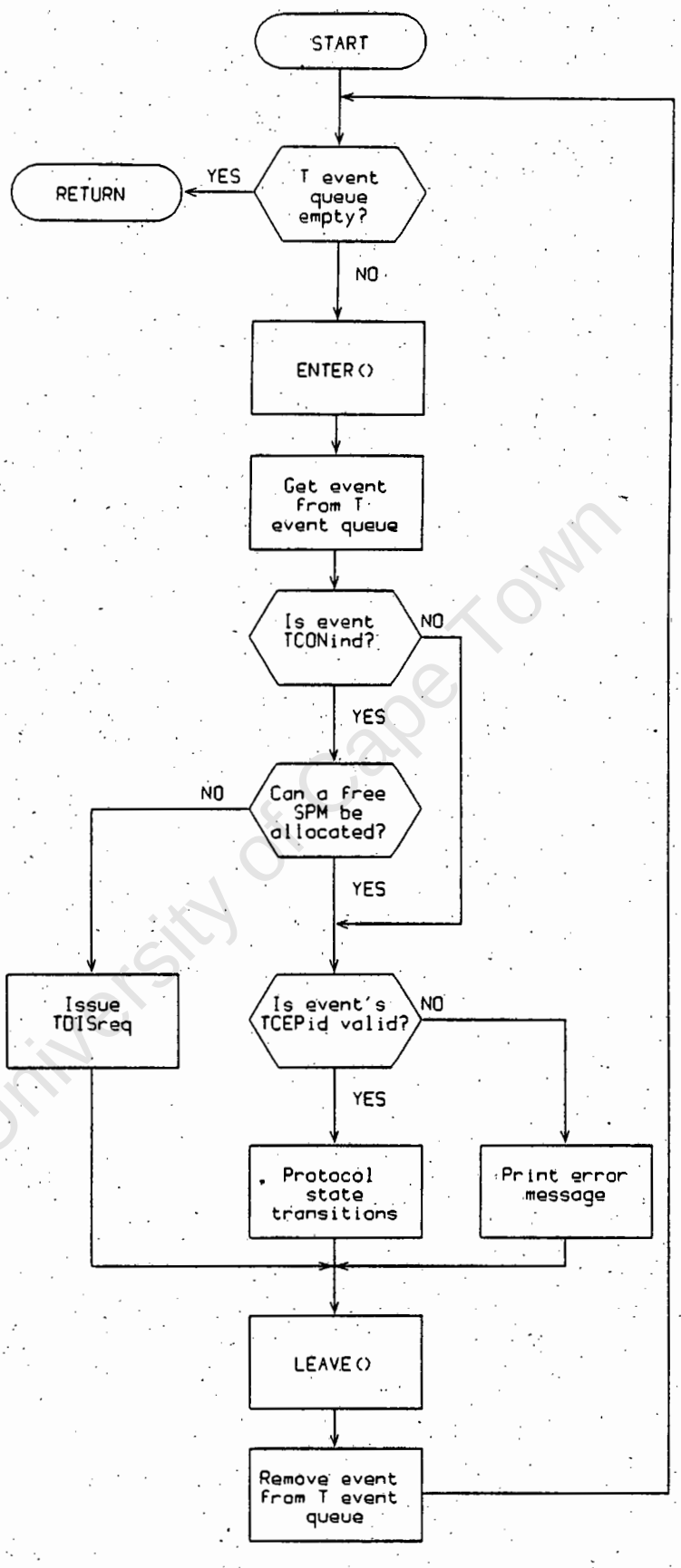

Figure 7.8 Flowchart for do_session_queue()

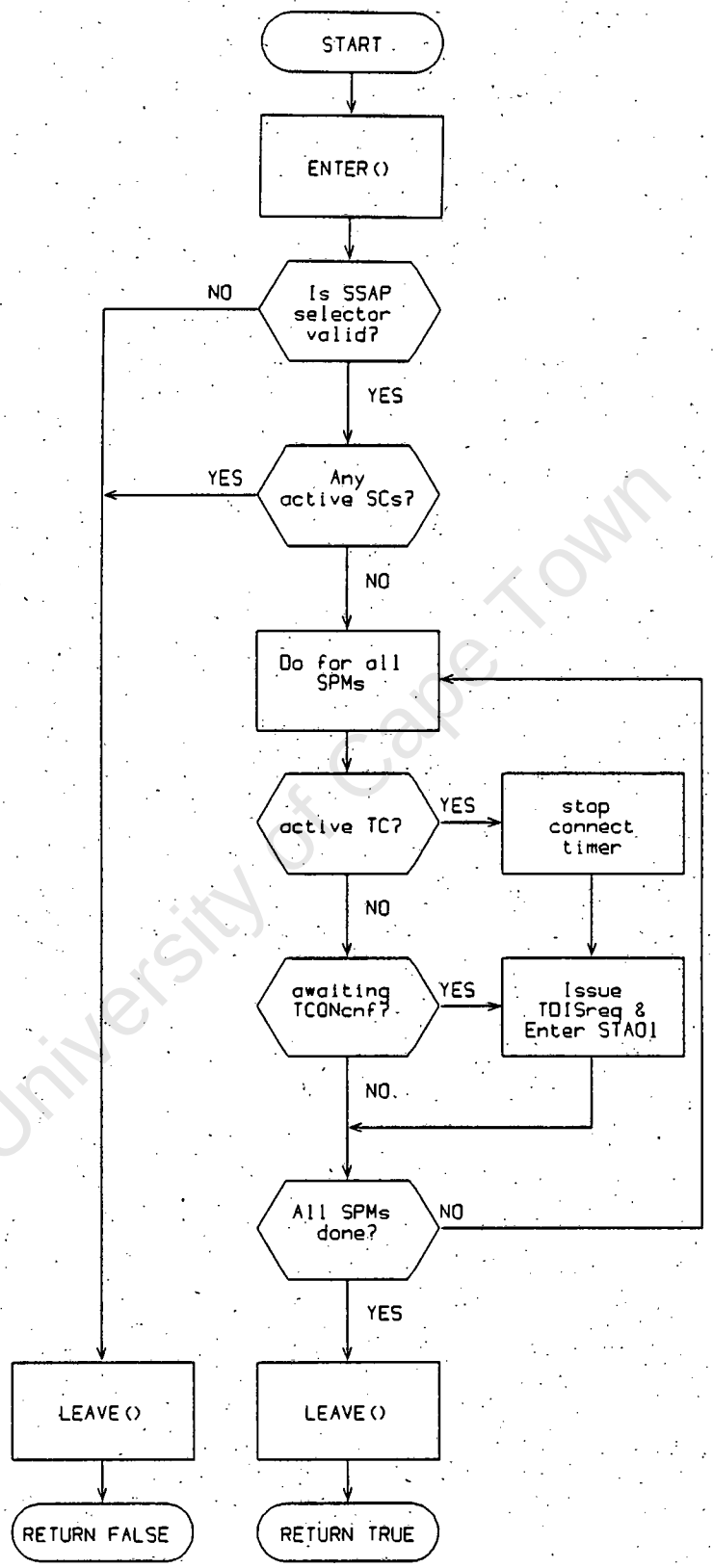

Figure 7.9 Flowchart for s_deactivate()

7.9 Implementation improvements

One aspect of this implementation that may need reviewing is the static array of SPMs that the session entity maintains in memory. The session entity allocates SPMs from this array to session and transport connections as the need arises. When not actively supporting a connection, these static structures represent a waste of memory.

In this implementation, this memory waste is not serious because of the relatively few (16) SPMs involved and because the system has lots of memory available for applications. Problems may arise if the session entity were to support a significantly greater number of SPMs, or if the system had less memory available for applications, or both.

In such cases it would be far better to use a dynamic memory allocation scheme for SPMs. This involves allocating memory for, and creating, a SPM only when specifically required. This SPM is then placed in a circular list of active SPMs. Once this SPM no longer supports an active session or transport connection, it is removed from the list and its memory is released.

7.10 Testing the software

In order to test the session entity software, it was decided to simulate actual operating conditions by running two concurrent RTS processes on the same system and have them communicate via their supporting session entities.

7.10.1 The pseudo transport layer

In order for the session entities to communicate, they need a transport connection between themselves. To provide this transport connection, a simple transport layer was designed to link the two session entities. It provides each session entity with one TSAP and supports one "transport connection" between them.

Because this transport layer resides only in one system, all that it has to do is to receive a transport request or response primitive from one session entity, convert it to its corresponding transport indication or confirm primitive, and deliver it to the other session entity.

This transport layer was implemented as a simple transport entity which interfaces to the session entity as part of the RTS module. An instance of this transport entity communicates in full-duplex fashion with its identical peer by means of an inter-process message queue managed by the UNIX inter-process communication facility.

The interface between the session and transport entity is fully conformant with that of subsection 7.6, except for one minor matter: in order for the session entity to receive incoming transport primitives, it must regularly call the following transport entity function:

do_transport_queue()

This function sequentially removes each incoming transport primitive from the inter-process message queue and calls the appropriate session entity transport indication and confirm primitive handlers. It returns once this queue is empty. In practice, the session entity calls this function as the first action in an RTS call to **do_session_queue**, since this is when the RTS expects incoming transport primitives to be processed.

Apart from this minor matter, the pseudo transport layer is totally transparent to the session entities. They (and their RTSs) may just as well be communicating with their peers in remote systems.

It is beyond the scope of this thesis to describe the implementation details of this transport entity. For these details, the reader is referred to the listing of the transport entity source file **tp2/transport.c**.

7.10.2 Monitoring the session entity

In order to monitor the operation of the session entity, a number of debug print statements were inserted at strategic places within the session entity code.

These statements print vital information regarding every function call across the session and transport interfaces. For every session entity state transition, they identify the input event, the initial state, the SPM being accessed, the output events, the final state and the function return value, if any. In addition, incoming and outgoing SPDUs are identified and their contents are printed as a series of decimal digits.

The information provided by these statements gives a complete, concise record of every step in the session entity's operation.

These debug statements are enabled by defining the C compiler directive **DEBUG** in the session entity's **makefile**. If debugging is enabled, the session entity source file **bas/session.c** includes a file of various string constants and functions used by the debug statements. These are contained in the source file **bas/debug.c**.

7.10.3 The test configuration

The RTS executable file was generated by running the **makefile** in the X.400 product's root directory. This **makefile** automatically runs the session entity and transport entity **makefiles** to compile their source files, and then links this object code with that of the RTS.

Two of these RTS processes were then run concurrently in the background, with their outputs (actually, the session entity debug outputs) re-directed to two separate files. A sample message was then placed in the outgoing message queue of one RTS, causing it to attempt session connection establishment with its peer. Figure 7.10 depicts this test configuration.

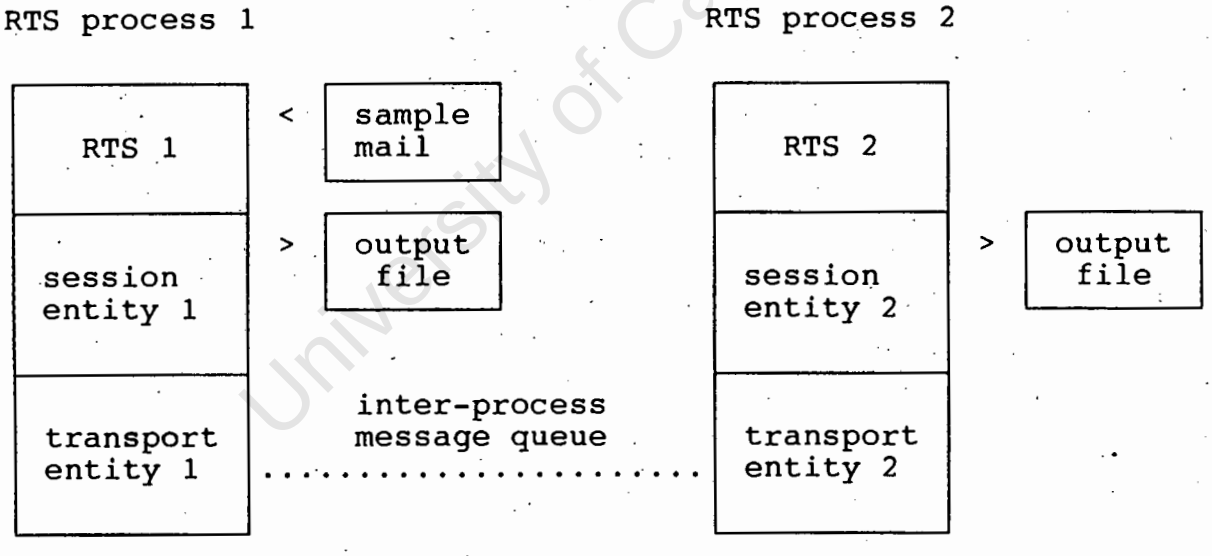

Figure 7.10 The test configuration

As shown in Figure 7.10, RTS 1 is the *calling RTS* and session entity 1 is the *initiator*. RTS 2 is the *called RTS* and session entity 2 is the *responder*.

7.10.4 Test results

The correspondent session entities were subjected to two separate test cases:

Test 1: Successful Session Connection Establishment,
Test 2: Unsuccessful Session Connection Establishment.

These two tests and their results are discussed separately below:

Test 1: Successful Session Connection Establishment:

In this test, the RTSs were allowed to proceed normally with the session connection establishment phase. Once a session connection had successfully been established, Session Entity 1 simply ignored all incoming data transfer phase primitives from RTS 1 until it received the S-ACTIVITY-END.request. It then simulated a TS-provider transport connection release by issuing a T-DISCONNECT.indication to itself and by issuing a T-DISCONNECT.request to its peer.

The debug outputs of the two session entities are listed in APPENDIX G.1. There is no need to explain these outputs here since they are self-explanatory. To aid the reader, each function call recorded in these two files has been sequentially numbered to clarify the sequence and interleaving of operations. This data shows that the session entity performs the session connection establishment phase correctly.

Test 2: Unsuccessful Session Connection Establishment:

In this test, a protocol error situation was simulated which would cause the called RTS to reject the session connection establishment attempt.

To simulate this protocol error, the proposed functional units from RTS 1 are intercepted by session entity 2 and indicated to RTS 2 as "no functional units required" in the S-CONNECT.indication. Naturally, RTS 2 cannot accept such a proposal and therefore aborts the connection attempt by issuing an S-U-ABORT.request.

Because the session entities support the reuse of transport connections, they exchange the required ABORT and ABORT ACCEPT SPDUs and enter state STA01C - idle, transport connection active. In doing so, they both start their connect timers. Because RTS 1 does not attempt another connection establishment, both these timers eventually expire, causing both session entities to release the transport connection.

The debug outputs of the two session entities are listed in APPENDIX G.2. As with the first test, the actions are sequentially numbered to show their interleaving and sequence. This data shows that the session entity performs connection abort and transport connection reuse correctly.

7.11 Alternative implementation strategies

The implementation technique described in this section has one drawback: because the OSI layer entities are implemented as processes which run concurrently with the host's applications under the same operating system, they compete with them for system resources. This problem is compounded when one considers running several RTS instances simultaneously. In this case, each RTS instance will require a complete session entity instance to support it. Running all these processes will result in significant overheads for the processor (mainly due to task swapping) and consequent degradation of the host's OSI throughput.

One solution might involve separating the session and RTS processes and then configuring one session entity to support several RTSs. This will produce significant savings on the sizes of the different processes and, consequently, processor overheads. However, the problem still remains that OSI layer entities compete with applications for system resources.

A solution to the latter problem might be to implement the OSI layer entities as a permanent part of the host operating system. From here, they may provide OSI services directly to the host's applications via, say, a library of system calls.

However, incorporating new services into an existing operating system is a very delicate problem. A better implementation technique would be to implement each different OSI subsystem on a separate processor with its own memory. This will avoid the problem of OSI subsystems competing with the host's applications for the host's resources while dramatically improving the host's OSI throughput. These may be very important considerations for certain applications.

It is worth pointing out that in many systems the transport layer is often implemented by a part of the host operating system. The network layer is typically implemented by an

input/output driver while the data link and physical layers are normally implemented in hardware. This illustrates that OSI subsystems may have to be implemented using a hybrid combination of different strategies.

8. CONCLUSIONS

The objectives of this thesis have been satisfied:

1) It has shown how the general session layer may be tailored to meet only the requirements of X.400.

   The RTS has been identified as the X.400 element that interacts directly with the session layer. Its minimal session service requirements have been specified and its use of these services has been described. The minimal session protocol required to provide only these services has been specified.

2) A formal description of this session layer has been presented.

   The session layer for X.400 has been formally specified using the Formal Description Technique Estelle. This specification includes a complete, minimal session entity capable of supporting the RTS.

3) It has shown how this session layer may be implemented and interfaced to an X.400 application on a real system.

   The session entity of the formal description has been partially implemented and interfaced to an existing X.400 product on a real system. This implementation uses the C programming language in a UNIX operating system environment.

References

1. CCITT, *Recommendations X.400 - X.430, Data Communication Networks, Message Handling Systems,* Red Book, Volume VIII, Fascicle VIII.7, Geneva, 1985.

2. CCITT, *Recommendation X.200, Reference Model of Open Systems Interconnection for CCITT Applications,* Red Book, Volume VIII, Fascicle VIII.5, Geneva, 1985.

3. CCITT, *Recommendation X.215, Session Service Definition for Open Systems Interconnection for CCITT Applications*, Red Book, Volume VIII, Fascicle VIII.5, Geneva, 1985.

4. CCITT, *Recommendation X.225, Session Protocol Specification for Open Systems Interconnection for CCITT Applications*, Red Book, Volume VIII, Fascicle VIII.5, Geneva, 1985.

5. CCITT, *Recommendation X.210, Open System Interconnection (OSI) Layer Service Definition Conventions*, Red Book, Volume VIII, Fascicle VIII.5, Geneva, 1985.

6. ISO, *Estelle: A Formal Description Technique Based on an Extended State Transition Model*, ISO/TC 97/SC 21 N xxxx/DP9074, September 1986.

7. Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, Englewood Cliffs, New Jersey: Prentice-Hall, 1978.

8. Rebecca Thomas and Jean Yates, *A User Guide to the UNIX System*, 2nd ed., Berkeley, California: Osborne McGraw-Hill, 1985.

9. CCITT, *Recommendation X.410, Message Handling Systems: Remote Operations and Reliable Transfer Server*, Red Book, Volume VIII, Fascicle VIII.7, Geneva, 1985.

10. CCITT, *Recommendation X.214, Transport Service Definition for Open Systems Interconnection (OSI) for CCITT Applications*, Red Book, Volume VIII, Fascicle VIII.5, Geneva, 1985.

11. CCITT, *Recommendation X.224, Transport Protocol Specification for Open Systems Interconnection for CCITT Applications*, Red Book, Volume VIII, Fascicle VIII.5, Geneva, 1985.

12. CCITT, *Recommendation Z.200, CCITT High Level Language (CHILL)*, Red Book, Volume VI, Fascicle VI.12, Geneva, 1985.

13. G. J. Dickson and G. R. Wheeler, *A Comparison of Formal Description Techniques Proposed for International Standardization*, Telecom Australia Research Laboratories, 770 Blackburn Road, Clayton, Victoria, Australia, 1984.

14. Gregor V. Bochmann and Carl A. Sunshine, "A Survey of Formal Methods" in *Computer Network Architectures and Protocols*, ed. Paul E. Green, Jr., New York: Plenum Press, 1982, pp.561-578.

15. CCITT, *Recommendations Z.101 - Z.104, Functional Specification and Description Language (SDL)*, Red Book, Volume VI, Fascicle VI.10, Geneva, 1985.

16. CCITT, *annexes to Recommendations Z.101 - Z.104, Functional Specification and Description Language (SDL)*, Red Book, Volume VI, Fascicle VI.11, Geneva, 1985.

17. R. Saracco and P. A. J. Tilanus, "CCITT SDL: Overview of the Language and its Applications", *Computer Networks and ISDN Systems*, No. 13, 1987, pp.65-74.

18. S. Budkowski and P. Dembinsky, "An Introduction to Estelle: A Specification Language for Distributed Systems", *Computer Networks and ISDN Systems*, No. 14, 1987, pp.3-23.

19. Richard J. Linn Jr., *A Tutorial on the Features and Facilities of Estelle*, National Bureau of Standards, Institute for Computer Science and Technology, Systems and Network Architecture Division, Gaithersburg, MD 20899, U.S.A., February 1986.

Bibliography

1. Black, U.D. *Data Communications and Distributed Networks*. 2nd ed. Englewood Cliffs, New Jersey: Prentice-Hall. 1987.

2. Bochmann, Gregor V. and Carl A. Sunshine. "A Survey of Formal Methods" in *Computer Network Architectures and Protocols*. ed. Paul E. Green, Jr. New York: Plenum Press. 1982. pp.561-578.

3. Brand, Daniel and Pitro Zafiropulo. "On Communicating Finite-State Machines". *Journal of the Association for Computing Machinery*. Vol. 30. No. 2. April 1983. pp.323-342.

4. Budkowski, S. and P. Dembinsky. "An Introduction to Estelle: A Specification Language for Distributed Systems". *Computer Networks and ISDN Systems*. No. 14. 1987. pp.3-23

5. Bunn, Christopher R. "Message Handling Systems and their Protocols". *Open Systems Data Transfer*. Transmission No. 16. June 1985. pp.1-28.

6. CCITT. *Recommendation X.200. Reference Model of Open Systems Interconnection for CCITT Applications*. Red Book. Volume VIII. Fascicle VIII.5. Geneva. 1985.

7. CCITT. *Recommendation X.210. Open System Interconnection (OSI) Layer Service Definition Conventions*. Red Book. Volume VIII. Fascicle VIII.5. Geneva. 1985.

8. CCITT. *Recommendation X.214. Transport Service Definition for Open Systems Interconnection for CCITT Applications*. Red Book. Volume VIII. Fascicle VIII.5. Geneva. 1985.

9. CCITT. *Recommendation X.215. Session Service Definition for Open Systems Interconnection for CCITT Applications*. Red Book. Volume VIII. Fascicle VIII.5. Geneva. 1985.

10. CCITT. *Recommendation X.225. Session Protocol Specification for Open Systems Interconnection for CCITT Applications*. Red Book. Volume VIII. Fascicle VIII.5. Geneva. 1985.

11. CCITT. *Recommendation X.400. Message Handling Systems: System Model-Service Elements*. Red Book. Volume VIII. Fascicle VIII.7. Geneva. 1985.

12. CCITT. *Recommendation X.410. Message Handling Systems: Remote Operations and Reliable Transfer Server*. Red Book. Volume VIII. Fascicle VIII.7. Geneva. 1985.

13. Cole, Robert. *Computer Communications*. 2nd ed. Houndmills: Macmillan. 1986.

14. Dickson, G. J. and G. R. Wheeler. *A Comparison of Formal Description Techniques Proposed for International Standardization*. Telecom Australia Research Laboratories. 770 Blackburn Road. Clayton. Victoria. Australia. 1984.

15. Exner, Rolf. "Protocols for Message Handling". *Australian Telecommunications Review*. Vol. 19. No. 1. 1985. pp.43-51

16. Halsall, F. *Introduction to Data Communications and Computer Networks*. Wokingham, England; Reading, Mass: 1985.

17. INTERACTIVE Systems Corporation. *AT&T UNIX V.3.2 Programmer's Reference Manual*. INTERACTIVE Systems Corporation. 2401 Colorado Avenue. 3rd Floor. Santa Monica. California 90404. 1988.

18. INTERACTIVE Systems Corporation. *AT&T UNIX V.3.2 Programmer's Guide Volumes I & II*. INTERACTIVE Systems Corporation. 2401 Colorado Avenue. 3rd Floor. Santa Monica. California 90404. 1988.

19. ISO. *Estelle: A Formal Description Technique Based on an Extended State Transition Model*. ISO/TC 97/SC 21 N xxxx/DP9074. September 1986.

20. Kernighan, Brian W. and Dennis M. Ritchie. *The C Programming Language*. Englewood Cliffs, New Jersey: Prentice-Hall. 1978.

21. Knightson, Keith G., Terry Knowles and John Larmouth. *Standards for Open Systems Interconnection*. New York: McGraw-Hill. 1987.

22. Linn, Richard J. Jr. *A Tutorial on the Features and Facilities of Estelle*. National Bureau of Standards. Institute for Computer Science and Technology. Systems and Network Architecture Division. Gaithersburg. MD 20899. U.S.A. February 1986.

23. Meijer, Anton and Paul Peeters. *Computer Network Architectures*. London: Pitman. 1982.

24. Saracco, R. and P. A. J. Tilanus. "CCITT SDL: Overview of the Language and its Applications". *Computer Networks and ISDN Systems*. No. 13. 1987. pp.65-74

25. Tanenbaum, Andrew S. *Computer Networks*. Englewood Cliffs, New Jersey: Prentice-Hall. 1981.

26. Thomas, Rebecca and Jean Yates. *A User Guide to the UNIX System*. 2nd ed. Berkeley, California: Osborne McGraw-Hill. 1985.

APPENDIX A. Session Service State Tables for the RTS

This appendix describes the session service as used by the RTS in terms of state tables. It shows how these are derived from those of CCITT Recommendation X.215 ANNEX A, which describes the general session service in terms of state tables. The state tables of this appendix use the notation, conventions and definitions established in CCITT Recommendation X.215 ANNEX A. These issues will therefore not be repeated here and a thorough knowledge of them will be assumed.

The RTS state tables are derived from the general state tables by extracting from the latter only those elements used by the RTS. This is achieved by simply omitting all those elements of the general state tables which are not used by the RTS. This process consists of the following eight sequential steps:

1) CCITT Recommendation X.215 ANNEX A.4.1 and A.4.2.2 state that the actions taken by the SS-user on detection of either an invalid intersection or a conditional action list for which none of the predicate expressions are true are local SS-user matters. These are therefore beyond the scope of this thesis.

2) Certain of the sets and variables defined in CCITT Recommendation X.215 ANNEX A.5 are not required by the RTS and may therefore be omitted. These are:

   a) From A.5.3:
      The subset of tokens **GT = {tokens given in the input event}** may be omitted because it is used only by the Give Tokens service, which is not used by the RTS.

b) From A.5.4.2:
The variables **Vrsp** and **Vrspnb** may be omitted because they are used only by the Resynchronization service, which is not used by the RTS.

c) From A.5.4.3:
The variable **Vcoll** may be omitted because it is used only when release requests collide. This can never happen to RTSs because only the sending RTS may request connection release.

d) From A.5.4.6:
The variable **V(R)** may be omitted because it is used only by the Resynchronization service, which is not used by the RTS.

3) All state table rows representing incoming SS-provider events (as defined in CCITT Recommendation X.215 TABLE A-1/X.215) associated with services not used by the RTS are omitted. These events are:

abbreviated name	name and description
SCDind	S-CAPABILITY-DATA.indication
SCDcnf	S-CAPABILITY-DATA.confirm
SEXind	S-EXPEDITED-DATA.indication
SGTind	S-TOKEN-GIVE.indication
SPERind	S-P-EXCEPTION-REPORT.indication
SRELcnf-	R-RELEASE.confirm (reject)
SRSYNind	S-RESYNCHRONIZE.indication
SRSYNcnf	S-RESYNCHRONIZE.confirm
SSYNMind	S-SYNC-MAJOR.indication
SSYNMcnf	S-SYNC-MAJOR.confirm
STDind	S-TYPED-DATA.indication

4) All state table rows representing outgoing SS-user events (as defined in CCITT Recommendation X.215 TABLE A-3/X.215) associated with services not used by the RTS are omitted. These events are:

abbreviated name	name and description
SCDreq	S-CAPABILITY-DATA.request
SCDrsp	S-CAPABILITY-DATA.response
SEXreq	S-EXPEDITED-DATA.request
SGTreq	S-TOKEN-GIVE.request
SRELrsp-	S-RELEASE.response (reject)
SRSYNreq	S-RESYNCHRONIZE.request
SRSYNrsp	S-RESYNCHRONIZE.response
SSYNMreq	S-SYNC-MAJOR.request
SSYNMrsp	S-SYNC-MAJOR.response
STDreq	S-TYPED-DATA.request

5) All state table columns representing states (as defined in CCITT Recommendation X.215 TABLE A-2/X.215) associated with services not used by the RTS are omitted. These states are:

abbreviated name	name and description
STA04A	await S-SYNC-MAJOR.confirm
STA05A	await S-RESYNCHRONIZE.confirm
STA10A	await S-SYNC-MAJOR.response
STA11A	await S-RESYNCHRONIZE.response
STA21	await S-CAPABILITY-DATA.confirm
STA22	await S-CAPABILITY-DATA.response

6) Of the remaining state table intersections, some have conditional action lists for which the boolean predicate conditions (as defined in CCITT Recommendation X.215 TABLE A-6/X.215) will never be true for RTS use. These actions lists will therefore never be performed by the RTS and may be omitted. These intersections are:

From TABLE A-8/X.215, Data transfer state table:

a) Intersection between STA09 (await SRELrsp) and SDTreq:
The predicate p04 (FU(FD) & ^Vcoll) will never be true because the RTS does not select the FD functional unit.

b) Intersection between STA10B (await SACTErsp) and SDTreq:
The predicate p03 (I(dk)) will never be true because, in order to enter STA10B, the RTS must not own the tokens.

c) Intersection between STA03 (await SRELcnf) and SDTind:
Intersection between STA04B (await SACTEcnf) and SDTind:
In order to enter these states, the RTS must own the tokens, preventing the remote RTS from issuing a SDTreq.

From TABLE A-11/X.215, Activity interrupt and discard state table:

a) Intersection between STA20 (await recovery request) and SACTDind:
Intersection between STA20 (await recovery request) and SACTIind:
To enter STA20 the RTS must own the tokens, which prevents the remote RTS from initiating any activity management services.

From TABLE A-13/X.215, Token management and exceptions state table:

a) Intersection between STA19 (await recovery indication) and SUERind:
To enter STA19 the RTS must not own the tokens, which prevents the remote RTS from issuing a SUERreq.

Table B.6 (part 3 of 3)

Token management and exceptions state table

INCOMING EVENT	CURRENT STATE				
	STA16 await TDISind	STA18 await GTA	STA19 await recovery (init)	STA20 await recovery	STA713 data transfer
ED	STA16				p51 SUERind STA20
GTA	STA16	STA713			
GTC	STA16				p62 SCGind GTA [11] STA713
PT	STA16	p53 SPTind STA18	p53 STA19	p53 STA20	p53 SPTind STA713
SCGreq					p55 GTC [11] STA18
SPTreq					p53 PT STA713
SUERreq					p50 ED STA19

Table B.7 (part 1 of 2)

Connection release state table

INCOMING EVENT	CURRENT STATE			
	STA01A await AA	STA01C idle, TC con	STA03 await DN	STA09 await SRELrsp
DN	STA01A	TDISreq STA01	^p66 SRELcnf+ TDISreq STA01 p66 SRELcnf+ STA01C	
FNnr	STA01A	TDISreq STA01		
FNr	STA01A	TDISreq STA01		
SRELreq				
SRELrsp+				^p66 DN [4] STA16 p66 DN STA01C

Table B.7 (part 2 of 2)

Connection release state table

INCOMING EVENT	CURRENT STATE		
	STA16 await TDISind	STA19 await recovery (init)	STA713 data transfer
DN	STA16		
FNnr	STA16	p68 STA19	p68 SRELind [8] STA09
FNr	STA16	p68&^p01 &p16 STA19	p68&^p01 &p16 SRELind [9] STA09
SRELreq			p63&^p64 FNnr [8] STA03 p63&p64 FNr [7] STA03
SRELrsp+			

Table B.8 (part 1 of 4)

Abort state table

INCOMING EVENT	CURRENT STATE				
	STA01 idle, no TC	STA01A await AA	STA01B await TCONcnf	STA01C idle, TC con	STA02A await AC
AA	//	[3] STA01C	//	TDISreq STA01	
ABnr	//	[3] TDISreq STA01	//	TDISreq STA01	SxABind TDISreq STA01
ABr	//	[3] STA01C	//	^p02 TDISreq STA01 p02 AA STA01C	^p02 SxABind TDISreq STA01 p02 SxABind AA STA01C
SUABreq			TDISreq STA01		^p02 ABnr [4] STA16 p02 ABr [4] STA01A
TDISind	//	[3] STA01	SPABind STA01	STA01	SPABind STA01
TIM	//	TDISreq STA01	//	//	//

Note: In Table B.8, SxABind means generate the event SUABind if bit 2 of the Transport Disconnect parameter in the ABORT SPDU has the value "user abort". Otherwise, generate the event SPABind.

Table B.8 (part 2 of 4)

Abort state table

INCOMING EVENT	CURRENT STATE				
	STA03 await DN	STA04B await AEA	STA05B await AIA	STA05C await ADA	STA08 await SCONrsp
AA					
ABnr	SxABind TDISreq STA01	SxABind TDISreq STA01	SxABind TDISreq STA01	SxABind TDISreq STA01	SxABind TDISreq STA01
ABr	^p02 SxABind TDISreq STA01 p02 SxABind AA STA01C	^p02 SxABind TDISreq STA01 p02 SxABind AA STA01C	^p02 SxABind TDISreq STA01 p02 SxABind AA STA01C	^p02 SxABind TDISreq STA01 p02 SxABind AA STA01C	^p02 SxABind TDISreq STA01 p02 SxABind AA STA01C
SUABreq	^p02 ABnr [4] STA16 p02 ABr [4] STA01A	^p02 ABnr [4] STA16 p02 ABr [4] STA01A	^p02 ABnr [4] STA16 p02 ABr [4] STA01A	^p02 ABnr [4] STA16 p02 ABr [4] STA01A	^p02 ABnr [4] STA16 p02 ABr [4] STA01A
TDISind	SPABind STA01	SPABind STA01	SPABind STA01	SPABind STA01	SPABind STA01
TIM	//	//	//	//	//

Table B.8 (part 3 of 4)

Abort state table

INCOMING EVENT	CURRENT STATE				
	STA09 await SRELrsp	STA10B await SACTErsp	STA11B await SACTIrsp	STA11C await SACTDrsp	STA16 await TDISind
AA					[3] TDISreq STA01
ABnr	SxABind TDISreq STA01	SxABind TDISreq STA01	SxABind TDISreq STA01	SxABind TDISreq STA01	[3] TDISreq STA01
ABr	^p02 SxABind TDISreq STA01 p02 SxABind AA STA01C	^p02 SxABind TDISreq STA01 p02 SxABind AA STA01C	^p02 SxABind TDISreq STA01 p02 SxABind AA STA01C	^p02 SxABind TDISreq STA01 p02 SxABind AA STA01C	[3] TDISreq STA01
SUABreq	^p02 ABnr [4] STA16 p02 ABr [4] STA01A	^p02 ABnr [4] STA16 p02 ABr [4] STA01A	^p02 ABnr [4] STA16 p02 ABr [4] STA01A	^p02 ABnr [4] STA16 p02 ABr [4] STA01A	
TDISind	SPABind STA01	SPABind STA01	SPABind STA01	SPABind STA01	[3] STA01
TIM	//	//	//	//	TDISreq STA01

Table B.8 (part 4 of 4)

Abort state table

INCOMING EVENT	CURRENT STATE			
	STA18 await GTA	STA19 await recovery (init)	STA20 await recovery	STA713 data transfer
AA				
ABnr	SxABind TDISreq STA01	SxABind TDISreq STA01	SxABind TDISreq STA01	SxABind TDISreq STA01
ABr	^p02 SxABind TDISreq STA01 p02 SxABind AA STA01C	^p02 SxABind TDISreq STA01 p02 SxABind AA STA01C	^p02 SxABind TDISreq STA01 p02 SxABind AA STA01C	^p02 SxABind TDISreq STA01 p02 SxABind AA STA01C
SUABreq	^p02 ABnr [4] STA16 p02 ABr [4] STA01A	^p02 ABnr [4] STA16 p02 ABr [4] STA01A	^p02 ABnr [4] STA16 p02 ABr [4] STA01A	^p02 ABnr [4] STA16 p02 ABr [4] STA01A
TDISind	SPABind STA01	SPABind STA01	SPABind STA01	SPABind STA01
TIM	//	//	//	//

CCITT Recommendation X.225 ANNEX A.4.1 states that actions taken by the SPM on invalid intersections between states and:

SS-user events,

TS-provider events, and

timer events

are local matters. To cater for such intersections, the following state table has been designed for use by the X.400 SPM:

Table B.9 (part 1 of 2) Invalid intersection state table

INCOMING EVENT	CURRENT STATE								
	01	01A	01B	01C	02A	03	04B	05B	05C
SCONreq	*	2	2	*	2	2	2	2	2
SCONrsp	1	2	2	2	2	2	2	2	2
SDTreq	1	2	2	2	2	2	2	2	2
SPTreq	1	2	2	2	2	2	2	2	2
SCGreq	1	2	2	2	2	2	2	2	2
SSYNmreq	1	2	2	2	2	2	2	2	2
SSYNmrsp	1	2	2	2	2	2	2	2	2
SUERreq	1	2	2	2	2	2	2	2	2
SACTSreq	1	2	2	2	2	2	2	2	2
SACTRreq	1	2	2	2	2	2	2	2	2
SACTIreq	1	2	2	2	2	2	*	2	2
SACTIrsp	1	2	2	2	2	2	2	2	2
SACTDreq	1	2	2	2	2	2	*	2	2
SACTDrsp	1	2	2	2	2	2	2	2	2
SACTEreq	1	2	2	2	2	2	2	2	2
SACTErsp	1	2	2	2	2	2	2	2	2
SRELreq	1	2	2	2	2	2	2	2	2
SRELrsp	1	2	2	2	2	2	2	2	2
SUABreq	1	2	*	2	*	*	*	*	*
TCONind	*	5	5	5	5	5	5	5	5
TCONcnf	3	5	*	5	5	5	5	5	5
TDTind	3	*	3	*	*	*	*	*	*
TDISind	4	*	*	*	*	*	*	*	*
TIM	6	*	6	6	6	6	6	6	6

Table B.9 (part 2 of 2) Invalid intersection state table

INCOMING EVENT	CURRENT STATE									
	08	09	10B	11B	11C	16	18	19	20	713
SCONreq	2	2	2	2	2	2	2	2	2	2
SCONrsp	*	2	2	2	2	2	2	2	2	2
SDTreq	2	2	2	2	2	2	2	2	2	*
SPTreq	2	*	*	2	2	2	2	2	2	*
SCGreq	2	2	2	2	2	2	2	2	2	*
SSYNmreq	2	2	2	2	2	2	2	2	2	*
SSYNmrsp	2	*	*	2	2	2	2	2	2	*
SUERreq	2	*	*	2	2	2	2	2	2	*
SACTSreq	2	2	2	2	2	2	2	2	2	*
SACTRreq	2	2	2	2	2	2	2	2	2	*
SACTIreq	2	2	2	2	2	2	2	2	*	*
SACTIrsp	2	2	2	*	2	2	2	2	2	2
SACTDreq	2	2	2	2	2	2	2	2	*	*
SACTDrsp	2	2	2	2	*	2	2	2	2	2
SACTEreq	2	2	2	2	2	2	2	2	2	*
SACTErsp	2	2	*	2	2	2	2	2	2	2
SRELreq	2	2	2	2	2	2	2	2	2	*
SRELrsp	2	*	2	2	2	2	2	2	2	2
SUABreq	*	*	*	*	*	2	*	*	*	*
TCONind	5	5	5	5	5	5	5	5	5	5
TCONcnf	5	5	5	5	5	5	5	5	5	5
TDTind	*	*	*	*	*	*	*	*	*	*
TDISind	*	*	*	*	*	*	*	*	*	*
TIM	6	6	6	6	6	*	6	6	6	6

Key:

* : the intersection is valid.

1 - 6 : the intersection is invalid:

1 : a) issue S-P-ABORT.indication, indicating "Protocol Error";
b) enter STA01.

2 : a) issue S-P-ABORT.indication, indicating "Protocol Error";
b) send an ABORT SPDU;
c) start the timer, TIM;
d) enter STA16.

3 : a) issue T-DISCONNECT.request;
b) enter STA01.

4 : a) enter STA01.

5 : a) issue S-P-ABORT.indication, indicating "Transport Disconnect";
b) issue T-DISCONNECT.request;
c) enter STA01.

6 : take no action.

APPENDIX C. The Estelle Specification Listing

This appendix lists the Estelle specification of the session layer for X.400.

```
SPECIFICATION OSI_environment;

  DEFAULT INDIVIDUAL QUEUE;

  TIMESCALE SECONDS;


{**********************************
 declaration part for specification
 **********************************}


{------------------------------------------
 constant-definition-part for specification
 ------------------------------------------}

CONST

{
 Maximum number of SCEPs per SSAP.
 This is therefore the maximum number of simultaneous session
 connections that a session entity can support.
}

NSCEPS = ANY INTEGER;


{
 Maximum number of TCEPs per TSAP.
 This is therefore the maximum number of simultaneous transport
 connections that a session entity can use.
 Note: the mappings between SSAPs and TSAPs, and
                between session and transport connections
                are one-to-one for X.400.

NTCEPS = NSCEPS;


{
 maximum SSDU length = n1024 bytes,
 where n = RTS checkpointSize parameter.
 If checkpointSize = 0, n = unlimited.
}

MAXSSDULEN = 1024 * ANY INTEGER;
```

```
DT_HEADER_LEN = 7; {maximum byte-length of DT SPDU header}

CONCAT_GT_LEN = 3; {concatenated GT SPDU length}

{
 maximum TSDU length =
 maximum, unsegmented DATA TRANSFER SPDU length +
 concatenated GIVE TOKENS SPDU length
}

MAXTSDULEN = MAXSSDULEN + DT_HEADER_LEN + CONCAT_GT_LEN;
```

```
{---------------------------------------
 type-definition-part for specification
 ---------------------------------------}

TYPE

{general TYPEs}

ByteTYPE = 0..255; {one byte}

                    {token identifiers}
TokenTYPE = (DKT,   {data}
             MIT,   {minor-synchronize}
             MAT,   {major/activity}
             TRT);  {release}

                    {functional unit identifiers}
FUTYPE = (HD,       {half-duplex}
          FD,       {duplex}
          EX,       {expedited data}
          SY,       {minor synchronize}
          MA,       {major synchronize}
          RESYN,    {resynchronize}
          ACT,      {activity management}
          NR,       {negotiated release}
          CD,       {capability data}
          EXCEP,    {exceptions}
          TD);      {typed data}

TokenSetTYPE = SET OF TokenTYPE; {set of tokens}

FUsetTYPE    = SET OF FUTYPE;    {set of functional units}
```

```
{
 multi-byte data TYPEs

 d = ARRAY of Bytes (data)
 l = current number (length) of significant Bytes
}

Bytes512TYPE = RECORD
               l : INTEGER;
               d : ARRAY [1..512] OF ByteTYPE;
               END;

Bytes64TYPE  = RECORD
               l : INTEGER;
               d : ARRAY [1..64] OF ByteTYPE;
               END;

Bytes32TYPE  = RECORD
               l : INTEGER;
               d : ARRAY [1..32] OF ByteTYPE;
               END;

Bytes16TYPE  = RECORD
               l : INTEGER;
               d : ARRAY [1..16] OF ByteTYPE;
               END;

Bytes9TYPE   = RECORD
               l : INTEGER;
               d : ARRAY [1..9] OF ByteTYPE;
               END;

Bytes6TYPE   = RECORD
               l : INTEGER;
               d : ARRAY [1..6] OF ByteTYPE;
               END;

Bytes4TYPE   = RECORD
               l : INTEGER;
               d : ARRAY [1..4] OF ByteTYPE;
               END;
```

```
{TYPEs for session service primitives}


{SSDU data structure}

SSDUTYPE = RECORD
           l : INTEGER;                                {length}
           d : ARRAY [1..MAXSSDULEN] OF ByteTYPE; {data}
           END;


{Individual token assignments}

TokenAssignTYPE = (REQUESTOR_SIDE,
                   ACCEPTOR_SIDE,
                   ACCEPTOR_CHOOSES);


{Initial token assignments}

InitialTokensTYPE = ARRAY [TokenTYPE] OF TokenAssignTYPE;


{SyncType values for SSYNmreq/ind}

SyncTypeTYPE = (EXPLICIT,  {explicit confirmation required}
                OPTIONAL); {explicit confirmation not required}


{Reason values for SUER,SACTI,SACTD req/ind}

ErActReasonType = (SSU_UNSPECIFIED,
                   SSU_CONGESTED,
                   SEQUENCE_ERROR,
                   LOCAL_SSU_ERROR,
                   PROCEDURE_ERROR,
                   DEMAND_DK);


{Result values for SCONrsp}

SCONrspResultType = (ACCEPT,          {SCONrsp+}
                     SSU_UNSPECIFIED, {SCONrsp-}
                     SSU_CONGESTED,   {SCONrsp-}
                     SSU_SEE_DATA);   {SCONrsp-}
```

```
{Result values for SCONcnf}

                                                {primitive  from}
SCONcnfResultType = (ACCEPT,                    {SCONcnf+   SSU }
                     SSU_UNSPECIFIED,           {SCONcnf-   SSU }
                     SSU_CONGESTED,             {SCONcnf-   SSU }
                     SSU_SEE_DATA,              {SCONcnf-   SSU }
                     CALLED_SSAP_UNKNOWN,       {SCONcnf-   SSP }
                     CALLED_SSU_UNATTACHED,     {SCONcnf-   SSP }
                     SSP_CONGESTED,             {SCONcnf-   SSP }
                     SSP_UNSPECIFIED);          {SCONcnf-   SSP }

{Result values for SRELrsp/cnf}

SRELresultTYPE = (AFFIRMATIVE, {SRELrsp+/cnf+}
                  NEGATIVE);   {SRELrsp-/cnf-}


{Reason values for SPABind}

SPABreasonTYPE = (TRANSPORT_DISCONNECT,
                  PROTOCOL_ERROR,
                  UNDEFINED);


{Quality Of Service: priority levels}

PriorityTYPE = 0..10;


{Quality Of Service: protection levels}

ProtectionTYPE = (LEVEL_A,  {no protection}
                  LEVEL_B,  {protection against monitoring}
                  LEVEL_C,  {protection against modification,
                             replay, addition and deletion}
                  LEVEL_D); {both B and C}
```

```
{QOSS parameter for SCONreq}

QOSSreqTYPE = RECORD

              Protection                 : ProtectionTYPE;
              Priority                   : PriorityTYPE;

              ResidualErrorRateDesired   : REAL;
              ResidualErrorRateMinimum   : REAL;

              Throughput0desired         : REAL;
              Throughput0minimum         : REAL;
              Throughput1desired         : REAL;
              Throughput1minimum         : REAL;

              TransitDelay0desired       : INTEGER;
              TransitDelay0minimum       : INTEGER;
              TransitDelay1desired       : INTEGER;
              TransitDelay1minimum       : INTEGER;

              ExtendedControl            : BOOLEAN;
              OptimizedDialogueTransfer  : BOOLEAN;

              END;


{QOSS parameter for SCONind/rsp/cnf}

QOSSTYPE    = RECORD

              Protection                 : ProtectionTYPE;
              Priority                   : PriorityTYPE;
              ResidualErrorRate          : REAL;

              Throughput0                : REAL;
              Throughput1                : REAL;

              TransitDelay0              : INTEGER;
              TransitDelay1              : INTEGER;

              ExtendedControl            : BOOLEAN;
              OptimizedDialogueTransfer  : BOOLEAN;

              END;
```

```
{TYPEs for transport service primitives}

{TSDU data structure}

TSDUTYPE = RECORD
           l : INTEGER;                                {length}
           i : INTEGER;                                {index}
           d : ARRAY [1..MAXTSDULEN] OF ByteTYPE; {data}
           END;

{Reason values for TDISind}

TDISreasonTYPE = (REMOTE_TSU_INVOKED,     {from TSU}
                  LACK_OF_RESOURCES,      {from TSP}
                  LOW_QOTS,               {from TSP}
                  TSP_MISBEHAVIOUR,       {from TSP}
                  CALLED_TSAP_UNKNOWN,    {from TSP}
                  CALLED_TSU_UNATTACHED, {from TSP}
                  UNKNOWN_REASON);        {from TSP}

{QOTS parameter}

QOTSTYPE = RECORD

           EstablishmentDelay              : INTEGER;
           EstablishmentFailureProbability : REAL;

           Throughput0maximum              : REAL;
           Throughput0average              : REAL;
           Throughput1maximum              : REAL;
           Throughput1average              : REAL;

           TransitDelay0maximum            : INTEGER;
           TransitDelay0average            : INTEGER;
           TransitDelay1maximum            : INTEGER;
           TransitDelay1average            : INTEGER;

           ResidualErrorRate               : REAL;
           TransferFailureProbability      : REAL;
           ReleaseDelay                    : INTEGER;
           ReleaseFailureProbability       : REAL;
           Protection                      : ProtectionTYPE;
           Priority                        : PriorityTYPE;
           Resilience                      : REAL;

           END;
```

```
{-----------------------------------
 channel-definition for specification
 -----------------------------------}

CHANNEL SSAPCHANNEL (SSuser,SSprovider);

        {SS primitives issued by X.400 SS-user (RTS)}

        BY SSuser:

           SCONreq(CallingSSuserRef : Bytes64TYPE;
                   CommonRef        : Bytes64TYPE;
                   AdditionalRef    : Bytes4TYPE;
                   CallingSSAPaddr  : Bytes16TYPE;
                   CalledSSAPaddr   : Bytes16TYPE;
                   qossReq          : QOSSreqTYPE;
                   Srequirements    : FUsetTYPE;
                   InitialSpsn      : INTEGER;
                   InitialTokens    : InitialTokensTYPE;
                   SSuserData       : Bytes512TYPE);

           SCONrsp(CalledSSuserRef  : Bytes64TYPE;
                   CommonRef        : Bytes64TYPE;
                   AdditionalRef    : Bytes4TYPE;
                   CalledSSAPaddr   : Bytes16TYPE;
                   Result           : SCONrspResultTYPE;
                   qoss             : QOSSTYPE;
                   Srequirements    : FUsetTYPE;
                   InitialSpsn      : INTEGER;
                   InitialTokens    : InitialTokensTYPE;
                   SSuserData       : Bytes512TYPE);

            SDTreq(ssdu             : SSDUTYPE);

            SPTreq(Tokens           : TokenSetTYPE;
                   SSuserData       : Bytes512TYPE);

            SCGreq;

          SSYNmreq(SyncType         : SyncTypeTYPE;
                   spsn             : INTEGER;
                   SSuserData       : Bytes512TYPE);

          SSYNmrsp(spsn             : INTEGER;
                   SSuserData       : Bytes512TYPE);

           SUERreq(Reason           : ErActReasonTYPE;
                   SSuserData       : Bytes512TYPE);

          SACTSreq(ActivityId       : Bytes6TYPE;
                   SSuserData       : Bytes512TYPE);
```

```
SACTRreq(ActivityId        : Bytes6TYPE;
         OldActivityId     : Bytes6TYPE;
         spsn              : INTEGER;
         CallingSSuserRef  : Bytes64TYPE;
         CalledSSuserRef   : Bytes64TYPE;
         CommonRef         : Bytes64TYPE;
         AdditionalRef     : Bytes4TYPE;
         SSuserData        : Bytes512TYPE);

SACTIreq(Reason            : ErActReasonTYPE);

SACTIrsp;

SACTDreq(Reason            : ErActReasonTYPE);

SACTDrsp;

SACTEreq(spsn              : INTEGER;
         SSuserData        : Bytes512TYPE);

SACTErsp(SSuserData        : Bytes512TYPE);

 SRELreq(SSuserData        : Bytes512TYPE);

 SRELrsp(Result            : SRELresultTYPE;
         SSuserData        : Bytes512TYPE);

 SUABreq(SSuserData        : Bytes9TYPE);
```

```
{SS primitives issued by SS-provider for X.400}

BY SSprovider:

   SCONind(CallingSSuserRef : Bytes64TYPE;
           CommonRef        : Bytes64TYPE;
           AdditionalRef    : Bytes4TYPE;
           CallingSSAPaddr  : Bytes16TYPE;
           CalledSSAPaddr   : Bytes16TYPE;
           qoss             : QOSSTYPE;
           Srequirements    : FUsetTYPE;
           InitialSpsn      : INTEGER;
           InitialTokens    : InitialTokensTYPE;
           SSuserData       : Bytes512TYPE);

   SCONcnf(CalledSSuserRef  : Bytes64TYPE;
           CommonRef        : Bytes64TYPE;
           AdditionalRef    : Bytes4TYPE;
           CalledSSAPaddr   : Bytes16TYPE;
           Result           : SCONcnfResultTYPE;
           qoss             : QOSSTYPE;
           Srequirements    : FUsetTYPE;
           InitialSpsn      : INTEGER;
           InitialTokens    : InitialTokensTYPE;
           SSuserData       : Bytes512TYPE);

    SDTind(ssdu             : SSDUTYPE);

    SPTind(Tokens           : TokenSetTYPE;
           SSuserData       : Bytes512TYPE);

    SCGind;

  SSYNmind(SyncType         : SyncTypeTYPE;
           spsn             : INTEGER;
           SSuserData       : Bytes512TYPE);

  SSYNmcnf(spsn             : INTEGER;
           SSuserData       : Bytes512TYPE);

   SUERind(Reason           : ErActReasonTYPE;
           SSuserData       : Bytes512TYPE);

  SACTSind(ActivityId       : Bytes6TYPE;
           SSuserData       : Bytes512TYPE);

  SACTRind(ActivityId       : Bytes6TYPE;
           OldActivityId    : Bytes6TYPE;
           spsn             : INTEGER;
           CallingSSuserRef : Bytes64TYPE;
           CalledSSuserRef  : Bytes64TYPE;
           CommonRef        : Bytes64TYPE;
           AdditionalRef    : Bytes4TYPE;
           SSuserData       : Bytes512TYPE);
```

```
SACTIind(Reason              : ErActReasonTYPE);

SACTIcnf;

SACTDind(Reason              : ErActReasonTYPE);

SACTDcnf;

SACTEind(spsn                : INTEGER;
         SSuserData          : Bytes512TYPE);

SACTEcnf(SSuserData          : Bytes512TYPE);

 SRELind(SSuserData          : Bytes512TYPE);

 SRELcnf(Result              : SRELresultTYPE;
         SSuserData          : Bytes512TYPE);

 SUABind(SSuserData          : Bytes9TYPE);

 SPABind(Reason              : SPABreasonTYPE);
```

```
CHANNEL TSAPCHANNEL (TSuser,TSprovider);

        {TS primitives issued by TS-user for X.400}

        BY TSuser:

           TCONreq(CallingTSAPaddr : Bytes16TYPE;
                   CalledTSAPaddr  : Bytes16TYPE;
                   ProposedTEXP    : BOOLEAN;
                   qots            : QOTSTYPE;
                   TSuserData      : Bytes32TYPE);

           TCONrsp(RespondTSAPaddr : Bytes16TYPE;
                   SelectedTEXP    : BOOLEAN;
                   qots            : QOTSTYPE;
                   TSuserData      : Bytes32TYPE);

            TDTreq(tsdu            : TSDUTYPE);

           TDISreq(TSuserData      : Bytes64TYPE);


        {TS primitives issued by TS-provider for X.400}

        BY TSprovider:

           TCONind(CallingTSAPaddr : Bytes16TYPE;
                   CalledTSAPaddr  : Bytes16TYPE;
                   ProposedTEXP    : BOOLEAN;
                   qots            : QOTSTYPE;
                   TSuserData      : Bytes32TYPE);

           TCONcnf(RespondTSAPaddr : Bytes16TYPE;
                   SelectedTEXP    : BOOLEAN;
                   qots            : QOTSTYPE;
                   TSuserData      : Bytes32TYPE);

            TDTind(tsdu            : TSDUTYPE);

           TDISind(Reason          : TDISreasonTYPE;
                   TSuserData      : Bytes64TYPE);
```

```
{-----------------------------------------
 module-header-definition for specificattion
 -----------------------------------------}

MODULE RTSmodule SYSTEMPROCESS;

  IP SSAP : ARRAY [1..NSCEPS] OF SSAPCHANNEL (SSuser);

  END;

MODULE SessionEntityModule SYSTEMPROCESS;

  IP SSAP : ARRAY [1..NSCEPS] OF SSAPCHANNEL (SSprovider);
     TSAP : ARRAY [1..NTCEPS] OF TSAPCHANNEL (TSuser);

  END;

MODULE TSproviderModule SYSTEMPROCESS;

  IP TSAPa,
     TSAPb : ARRAY [1..NTCEPS] OF TSAPCHANNEL (TSprovider);

  END;


{-----------------------------------------
 module-body-definition for specification
 -----------------------------------------}

BODY RTSbody          FOR RTSmodule;

  EXTERNAL; {Not specified by this specification.}

BODY SessionEntityBody FOR SessionEntityModule;

  EXTERNAL; {Still to be specified by this specification.}

BODY TSproviderBody    FOR TSproviderModule;

  EXTERNAL; {Not specified by this specification.}


{-------------------------------------------------
 interaction-point-declaration-part for specification
 -------------------------------------------------}

{Empty.
 The specification has no internal interaction points.}
```

```
{-----------------------------------------------
 module-variable-declaration-part for specification
{-----------------------------------------------}

MODVAR RTSinstanceA,
       RTSinstanceB            : RTSmodule;

       SessionEntityInstanceA,
       SessionEntityInstanceB  : SessionEntityModule;

       TSproviderInstance      : TSproviderModule;


{-----------------------------------------
 variable-declaration-part for specification
 -----------------------------------------}

{Empty.
 The specification has no variables.}


{---------------------------------------
 state-definition-part for specification
 ---------------------------------------}

{Empty.
 The specification does not need states because it is inactive.}


{-----------------------------------------
 state-set-definition-part for specification
 -----------------------------------------}

{Empty.
 The specification does not need states because it is inactive.}


{-------------------------------------------------------
 procedure-and-function-declaration-part for specification
 -------------------------------------------------------}

{Empty.
 The specification has no procedures or functions.}
```

```
{************************************
 initialization-part for specification
 ************************************}

{This part creates the static configuration of specification child
 module instances and the static links between them.}

INITIALIZE

  BEGIN

    {create two RTS module instances}

    INIT RTSinstanceA WITH RTSbody;
    INIT RTSinstanceB WITH RTSbody;


    {create two session entity module instances}

    INIT SessionEntityInstanceA WITH SessionEntityBody;
    INIT SessionEntityInstanceB WITH SessionEntityBody;


    {create one TSprovider module instance}

    INIT TSproviderInstance WITH TSproviderBody;


    {connect all SSAPs together to form SCEPs}

    ALL i : 1..NSCEPS DO
      BEGIN
        CONNECT RTSinstanceA.SSAP[i]
             TO SessionEntityInstanceA.SSAP[i];
        CONNECT RTSinstanceB.SSAP[i]
             TO SessionEntityInstanceB.SSAP[i];
      END;


    {connect all TSAPs together to form TCEPs}

    ALL i : 1..NTCEPS DO
      BEGIN
        CONNECT SessionEntityInstanceA.TSAP[i]
             TO TSproviderInstance.TSAPa[i];
        CONNECT SessionEntityInstanceB.TSAP[i]
             TO TSproviderInstance.TSAPb[i];
      END;

  END;
```

```
{*********************************************
 transition-declaration-part for specification
 *********************************************}

{Empty.
 The specification has no transitions because it is inactive.}
```

```
{
 SessionEntityBody module body definition,
 as nested by module-body-definition for specification.
}


{**************************************
 declaration part for SessionEntityBody
 **************************************}


{-----------------------------------------------
 constant-definition-part for SessionEntityBody
 -----------------------------------------------}

CONST

NSPMS = NSCEPS; {maximum number of SPMs, and therefore
                 simultaneous session connections}


{-----------------------------------------
 type-definition-part for SessionEntityBody
 -----------------------------------------}

{Empty.
 SessionEntityBody needs no additional types.}


{---------------------------------------
 channel-definition for SessionEntityBody
 ---------------------------------------}

{Empty.
 SessionEntityBody needs no additional channel types.}


{-----------------------------------------------
 module-header-definition for SessionEntityBody
 -----------------------------------------------}

MODULE SPMmodule PROCESS;

  IP SCEP : SSAPCHANNEL (SSprovider);
     TCEP : TSAPCHANNEL (TSuser);

  END;
```

```
{-------------------------------------------
 module-body-definition for SessionEntityBody
 -------------------------------------------}

BODY SPMbody FOR SPMmodule;

  EXTERNAL; {Still to be specified by this specification.}


{-----------------------------------------------------------
 interaction-point-declaration-part for SessionEntityBody
 -----------------------------------------------------------}

{Empty.
 SessionEntityBody has no internal interaction points.}


{-----------------------------------------------------
 module-variable-declaration-part for SessionEntityBody
 -----------------------------------------------------}

MODVAR SPMinstance : ARRAY [1..NSPMS] OF SPMmodule;


{-------------------------------------------------
 variable-declaration-part for SessionEntityBody
 -------------------------------------------------}

{Empty.
 SessionEntityBody has no variables.}


{-------------------------------------------
 state-definition-part for SessionEntityBody
 -------------------------------------------}

{Empty.
 SessionEntityBody does not need states because it is inactive.}


{-------------------------------------------------
 state-set-definition-part for SessionEntityBody
 -------------------------------------------------}

{Empty.
 SessionEntityBody does not need states because it is inactive.}
```

```
{-------------------------------------------------------------
 procedure-and-function-declaration-part for SessionEntityBody
 -------------------------------------------------------------}

{Empty.
 SessionEntityBody has no procedures or functions.}


{****************************************
 initialization-part for SessionEntityBody
 ****************************************}

INITIALIZE

  BEGIN

    {Create the static configuration of child SPM module
     instances and attach each between a SCEP/TCEP pair.}


    {create NSPMS SPM module instances}

    ALL i : 1..NSPMS DO
      INIT SPMinstance[i] WITH SPMbody;


    {attach each SPM to a SCEP and to a TCEP}

    ALL i : 1..NSPMS DO
      BEGIN
        ATTACH SPMinstance[i].SCEP
            TO SSAP[i];
        ATTACH SPMinstance[i].TCEP
            TO TSAP[i];
      END;

  END;


{************************************************
 transition-declaration-part for SessionEntityBody
 ************************************************}

{Empty.
 SessionEntityBody has no transitions because it is inactive.}
```

```
{
 SPMbody module body definition,
 as nested by module-body-definition for SessionEntityBody.
}


{****************************
 declaration-part for SPMbody
 ****************************}


{-----------------------------------
 constant-definition-part for SPMbody
 -----------------------------------}

CONST

VERSION     = 1;            {local session protocol version number}

PROTOCOL    = 0;            {local session protocol options:
                             extended concatenation not supported}

TEXP_LOCAL = FALSE;         {transport expedited data local option}

REUSE_TC    = TRUE;         {reuse TC local option}

PERIOD      = ANY INTEGER; {time period for timer TIM}

DEFAULT_SPSN = 0;           {default serial number}
```

```
{--------------------------------
 type-definition-part for SPMbody
 --------------------------------}


TYPE

{Enclosure Item parameter values for DT SPDU}

                                              {value}
EnclosureItemTYPE = (NOT_BEGIN_NOT_END,       {   0   }
                     BEGIN_NOT_END,           {   1   }
                     NOT_BEGIN_END,           {   2   }
                     BEGIN_END);              {   3   }


{
 Reason field values for ReasonCodeTYPE, i.e., values for the
 lst byte of the Reason Code parameter for RF ED AI AD SPDUs
}

                                        {    SPDUs      from  value}
ReasonTYPE = (SSU_UNSPECIFIED,          {RF ED AI AD   SSU      0 }
              SSU_CONGESTED,            {RF ED AI AD   SSU      1 }
              SSU_SEE_DATA,             {RF            SSU      2 }
              SEQUENCE_ERROR,           {   ED AI AD   SSU      3 }
              LOCAL_SSU_ERROR,          {   ED AI AD   SSU      5 }
              PROCEDURE_ERROR,          {   ED AI AD   SSU      6 }
              DEMAND_DK,                {   ED AI AD   SSU    128 }
              CALLED_SSAP_UNKNOWN,      {RF            SSP    129 }
              CALLED_SSU_UNATTACHED,    {RF            SSP    130 }
              SSP_CONGESTED,            {RF            SSP    131 }
              PROPOSED_PROTOCOL);       {RF            SSP    132 }


{Reason Code parameter for RF ED AI AD SPDUs}

ReasonCodeTYPE = RECORD
                   Reason : ReasonTYPE;   {byte  1}
                   Data   : Bytes512TYPE; {bytes 2-513}
                   END;
```

```
{
 ABreason field values for TCdisTYPE, i.e., values for bits 2-4
 of the Transport Disconnect parameter for RF AB FN SPDUs
}

                                  { SPDUs     abort by  value}
ABreasonTYPE = (NO_ABORT,         {RF FN      no abort    0  }
                USER_ABORT,       {       AB    SSU       2  }
                PROTOCOL_ERROR,   {       AB    SSP       4  }
                NO_REASON),       {       AB    SSP       8  }


{Transport Disconnect parameter for RF AB FN SPDUs}

TCdisTYPE = RECORD
            TCkept   : BOOLEAN;       {bit  1}
            ABreason : ABreasonTYPE; {bits 2-4}
            END;


{SPDU identifiers used by PROCEDURE idSPDU}

SPDUidTYPE  = ( CN, AC, DT, GT, PT, DN, AS,
                AI, AR, AD, AE, AA, ED,

              AIA,ADA,AEA,MIP,MIA,GTC,GTA,

                RF,RF_R,RF_NR,
                FN,FN_R,FN_NR,
                AB,AB_R,AB_NR);
```

```
{------------------------------
 channel-definition for SPMbody
 ------------------------------}

CHANNEL TIMSAPCHANNEL (TIMuser,TIMprovider);

        {Timer Service Primitives}

        BY TIMuser:

           START(period : INTEGER);

           STOP;

        BY TIMprovider:

           TIMEOUT;


{-------------------------------------
 module-header-definition for SPMbody
 -------------------------------------}

MODULE TimerModule ACTIVITY;

  IP TIMSAP : TIMSAPCHANNEL (TIMprovider);

  END;


{-----------------------------------
 module-body-definition for SPMbody
 -----------------------------------}

BODY TimerBody FOR TimerModule;

{declaration-part for TimerBody}

CONST HIGH = 1;
      LOW  = 2;

VAR   delay : INTEGER;

STATE IDLE,ACTIVE,RESTART;
```

```
{initialization-part for TimerBody}

INITIALIZE
  TO IDLE
    BEGIN
    END;


{transition-declaration-part for TimerBody}

TRANS

  FROM IDLE
    WHEN TIMSAP.START(period)
      TO ACTIVE
        BEGIN
          delay := period;
        END;

    WHEN TIMSAP.STOP
      TO SAME
        BEGIN
        END;


  FROM ACTIVE
    PRIORITY HIGH
      WHEN TIMSAP.START(period)
        TO RESTART
          BEGIN
            delay := period;
          END;

      WHEN TIMSAP.STOP
        TO IDLE
          BEGIN
          END;

    PRIORITY LOW
      TO IDLE
        DELAY(delay)
          BEGIN
            OUTPUT TIMSAP.TIMEOUT;
          END;


  FROM RESTART
    TO ACTIVE
      BEGIN
      END;
```

```
{-----------------------------------------
 interaction-point-declaration-part for SPMbody
{-----------------------------------------}

IP TIMSAP : TIMSAPCHANNEL (TIMuser);


{-----------------------------------------
 module-variable-declaration-part for SPMbody
 -----------------------------------------}

MODVAR TimerInstance : TimerModule;


{-------------------------------------
 variable-declaration-part for SPMbody
 -------------------------------------}

VAR

{
 Required variables.
 These are as specified in Rec. X.225 A.5.4.
}

Texp     : BOOLEAN;  {transport expedited data service selected}

Vact     : BOOLEAN;  {activity in progress}

Vnextact : BOOLEAN;  {next Vact when AEA SPDU sent or received}

Vtca     : BOOLEAN;  {transport connection acceptor}

Vtrr     : BOOLEAN;  {SPM may reuse transport connection}

Va       : INTEGER;  {lowest spsn for which confirmation expected}

Vm       : INTEGER;  {next spsn to be used}

Vsc      : BOOLEAN;  {right to issue SSYNmrsp when Va < Vm}
```

```
{
 Miscellaneous variables.
 These (except AssembleSSDU) are initialized during the
 Session Connection Establishment Phase.
}

SelectedFUs      : FUsetTYPE;     {selected functional units}

AvailableTokens : TokenSetTYPE; {available tokens}

OwnedTokens      : TokenSetTYPE; {owned tokens}

Protocol         : ByteTYPE;      {remote SPM protocol options}

Version          : ByteTYPE;      {selected version number}

SelectedQOSS     : QOSSTYPE;      {selected QOSS}

SelectedQOTS     : QOTSTYPE;      {selected QOTS}

RemoteAddress    : Bytes16TYPE;   {remote SSAP or TSAP address}

LocalAddress     : Bytes16TYPE;   {local  SSAP or TSAP address}

TempTSDU         : TSDUTYPE;      {temporary TSDU storage}

{selected maximum TSDU lengths}

MaxTSDU0    : INTEGER; {initiator to responder}

MaxTSDU1    : INTEGER; {responder to initiator}

TempTokens : InitialTokensTYPE; {proposed initial token positions}

{SSDU used for reassembling incoming, segmented SSDUs}

AssembleSSDU : SSDUTYPE;
```

```
{
 'Constant' variables.
 These are assigned constant values by the initialization-part.
 They cannot be defined in the constant-definition-part due to
 their TYPEs.
}

TK_DOM       : TokenSetTYPE;       {set of all tokens}

FU_DOM       : FUsetTYPE;          {set of all functional units}

FU_SUP       : FUsetTYPE;          {supported functional units}

DEFAULT_QOSS : QOSSTYPE;           {default QOSS values

DEFAULT_QOTS : QOTSTYPE;           {default QOTS values

DEFAULT_TKNS : InitialTokensTYPE; {default token assignments}


{---------------------------------
 state-definition-part for SPMbody
 ---------------------------------}


STATE

STA01,   {idle, no transport connection}
STA01A,  {await AA SPDU}
STA01B,  {await TCONcnf}
STA01C,  {idle, transport connection}
STA02A,  {await AC SPDU}
STA03 ,  {await DN SPDU}
STA04B,  {await AEA SPDU}
STA05B,  {await AIA SPDU}
STA05C,  {await ADA SPDU}
STA08 ,  {await SCONrsp}
STA09 ,  {await SRELrsp}
STA10B,  {await SACTErsp}
STA11B,  {await SACTIrsp}
STA11C,  {await SACTDrsp}
STA16 ,  {await TDISind}
STA18 ,  {await GTA SPDU}
STA19 ,  {await recovery request or SPDU (initiator of ED SPDU)}
STA20 ,  {await recover SPDU or request}
STA713;  {data transfer}


{-------------------------------------
 state-set-definition-part for SPMbody
 -------------------------------------}

{Empty.
 No state sets are defined for SPMbody.}
```

```
{
 procedure-and-function-declaration-part for SPMbody
}


{
 PROCEDURE: AppendTSDU

            This procedure appends the contents of a given source
            TSDU onto a given target TSDU. It is assumed that
            this concatenation will not result in a TSDU length
            overflow.

 INPUTS:    tsdu1 - the target TSDU. Its .l field indicates its
                    current length.

            tsdu2 - the source TSDU. Its .l field indicates its
                    length.

 OUTPUTS:   tsdu1.l - is updated to include the appended bytes
                      of tsdu2, if any.

 CALLS:     none.
}


PURE PROCEDURE AppendTSDU(VAR tsdu1 : TSDUTYPE;
                              tsdu2 : TSDUTYPE);

VAR i : INTEGER;

BEGIN
  IF tsdu2.l > 0
  THEN
    FOR i := 1 TO tsdu2.l DO
      tsdu1.d[tsdu1.l+i] := tsdu2.d[i];
  tsdu1.l := tsdu1.l + tsdu2.l;
END;
```

```
{
 PROCEDURE: AppendSSDU

            This procedure appends the contents of a given source
            SSDU onto a given target SSDU. It is assumed that
            this concatenation will not result in a SSDU length
            overflow.

 INPUTS:    ssdu1 - the target SSDU. Its .l field indicates its
                    current length.

            ssdu2 - the source SSDU. Its .l field indicates its
                    length.

 OUTPUTS:   ssdu1.l - is updated to include the appended bytes
                      of ssdu2, if any.

 CALLS:     none.
}


PURE PROCEDURE AppendSSDU(VAR ssdu1 : SSDUTYPE;
                              ssdu2 : SSDUTYPE);

VAR i : INTEGER;

BEGIN
  IF ssdu2.l > 0
  THEN
    FOR i := 1 TO ssdu2.l DO
      ssdu1.d[ssdu1.l+i] := ssdu2.d[i];
  ssdu1.l := ssdu1.l + ssdu2.l;
END;
```

```
{
 PROCEDURE: BuildHeader

            This procedure appends a PIU, PGIU or SPDU header, as
            part of a SPDU, onto a given TSDU.

 INPUTS:    tsdu  - the TSDU. Its .l field indicates its current
                    length.

            Pcode - the PI, PGI or SI code.

            LI    - the Length Indicator.

 OUTPUTS:   tsdu.l - is updated to include the appended header.

 CALLS:     none.
}


PURE PROCEDURE BuildHeader(VAR tsdu  : TSDUTYPE;
                               Pcode : ByteTYPE;
                               LI    : INTEGER);

BEGIN
  tsdu.d[tsdu.l+1] := Pcode;

  IF LI > 254
  THEN        {3 byte LI field}
    BEGIN
      tsdu.d[tsdu.l+2] := 255;
      tsdu.d[tsdu.l+3] := LI DIV 256;
      tsdu.d[tsdu.l+4] := LI MOD 256;
      tsdu.l := tsdu.l + 4;
    END;
  ELSE        {1 byte LI field}
    BEGIN
      tsdu.d[tsdu.l+2] := LI;
      tsdu.l := tsdu.l + 2;
    END;
END;
```

```
{
 PROCEDURE: StripHeader

            This procedure strips a PIU, PGIU or SPDU header, as
            part of a SPDU, from a given TSDU.

 INPUTS:    tsdu  - the TSDU. Its .l field indicates its current
                    length and its .i field points to the last
                    byte stripped.

            Pcode - the PI, PGI or SI code of the required
                    PIU, PGIU or SPDU. If Pcode is 0, the header
                    is stripped irrespective of the actual
                    PI, PGI or SI value.

            LI    - the variable to hold the Length Indicator of
                    the stripped header.

 OUTPUTS:   LI - the PIU, PGIU or SPDU LI value.
            LI will = 0 iff:
              a) end of tsdu has been reached, OR
              b) the PIU, PGIU or SPDU is not the required one, OR
              c) the LI field contains the value 0.
            In this event, there are no parameters for the
            required PIU, PGIU or SPDU present in tsdu.

            tsdu.i - is updated to exclude the stripped header,
                     if it was present.

 CALLS:     none.
}


PURE PROCEDURE StripHeader(VAR tsdu  : TSDUTYPE;
                               Pcode : ByteTYPE;
                           VAR LI    : INTEGER);

BEGIN
  LI := 0;                                {default LI value}
  IF (tsdu.i < tsdu.l) AND                {if not end of TSDU AND  }
     (tsdu.d[tsdu.i+1] = Pcode OR         {if required PI/PGI/SI OR}
                 Pcode = 0       )) {        any PI/PGI/SI   }
  THEN
    BEGIN
      LI := tsdu.d[tsdu.i+2];             {get 1 byte LI}
      tsdu.i := tsdu.i + 2;
      IF LI = 255                         {if 3 byte LI field}
      THEN                                {get 2 byte LI}
        BEGIN
          LI := ORD(tsdu.d[tsdu.i+1])*256 +
                ORD(tsdu.d[tsdu.i+2]);
          tsdu.i := tsdu.i + 2;
        END;
    END;
END;
```

```
{
 FUNCTION: bitAND
           This function returns the bit-wise AND of two given
           ByteTYPEs.

 INPUTS:   bytel, byte2 - the given ByteTYPEs.

 OUTPUTS:  returns: the bit-wise AND of bytel and byte2.

 CALLS:    none.
}


PURE FUNCTION bitAND(tsdul, tsdu2 : ByteTYPE) : ByteTYPE;

VAR result, weight, i : INTEGER;

BEGIN
  result := 0;
  weight := 128;

  FOR i := 8 DOWNTO 1 DO
    BEGIN

      IF ((ORD(bytel) DIV weight) = 1) AND
         ((ORD(byte2) DIV weight) = 1)
      THEN
          result := result + weight;

      bytel := ORD(bytel) MOD weight;
      byte2 := ORD(byte2) MOD weight;
      weight := weight DIV 2;
    END;

  bitAND := result;
END;
```

```
{
 FUNCTION: Functional Unit and Token functions

           These functions implement those specified in
           Rec. X.225 A.5.1 - A.5.3.

 function   calls

   FU       none
   AV       FU
   OWNED    none
   I        AV OWNED
   II       AV OWNED
   A        AV OWNED
   AA       AV OWNED
   ALLT     none
   ANYT     none
}


{
 FU(fu) = TRUE iff the functional unit fu was selected.
}

PURE FUNCTION FU(fu : FUTYPE) : BOOLEAN;

BEGIN
  FU := fu IN SelectedFUs;
END;


{
 AV(token) = TRUE iff token is available.
}

PURE FUNCTION AV(token : TokenTYPE) : BOOLEAN;

BEGIN
  CASE token OF
    mi: AV := FU(SY);
    dk: AV := FU(HD);
    tr: AV := FU(NR);
    ma: AV := FU(MA) OR FU(ACT);
  END;
END;
```

```
{
 OWNED(token) = TRUE iff token is owned.
}

PURE FUNCTION OWNED(token : TokenTYPE) : BOOLEAN;

BEGIN
  OWNED := token IN OwnedTokens;
END;


{
 I(token) = TRUE indicates that actions controlled by token may
 be initiated by this side even if token is not available.
}

PURE FUNCTION I(token : TokenTYPE) : BOOLEAN;

BEGIN
  I := NOT AV(token) OR OWNED(token);
END;


{
 II(t) = TRUE indicates that actions controlled by token may be
 initiated by this side iff token is available.
}

PURE FUNCTION II(token : TokenTYPE) : BOOLEAN;

BEGIN
  II := AV(token) AND OWNED(token);
END;


{
 A(token) = TRUE indicates that actions controlled by token may
 be accepted by this side even if token is not available.
}

PURE FUNCTION A(token : TokenTYPE) : BOOLEAN;

BEGIN
  A := NOT AV(token) OR NOT OWNED(token);
END;
```

```
{
 AA(token) = TRUE indicates that actions controlled by token
 may be accepted by this side iff token is available.
}

PURE FUNCTION AA(token : TokenTYPE) : BOOLEAN;

BEGIN
  AA := AV(token) AND NOT OWNED(token);
END;


{
 ALLT(TF(token),tset) = TRUE iff all TF(token) = TRUE for token
 in tset, or tset empty.
}

PURE FUNCTION ALLT(FUNCTION TF(token : TokenTYPE) : BOOLEAN;
                   tset : TokenSetTYPE) : BOOLEAN;

BEGIN
  FORONE token : TokenTYPE
    SUCHTHAT (token IN tset) AND NOT TF(token) DO
      ALLT := FALSE;
    OTHERWISE
      ALLT := TRUE;
END;


{
 ANYT(TF(token),tset) = TRUE iff any TF(token) = TRUE for
 token in tset.
}

PURE FUNCTION ANYT(FUNCTION TF(token : TokenTYPE) : BOOLEAN;
                   tset : TokenSetTYPE) : BOOLEAN;

BEGIN
  FORONE token : TokenTYPE
    SUCHTHAT (token IN tset) AND TF(token) DO
      ANYT := TRUE;
    OTHERWISE
      ANYT := FALSE;
END;
```

```
{
 FUNCTION: MapErActReq

           This function maps the Reason (ErActReasonTYPE)
           parameter value of the SUERreq, SACTIreq or SACTDreq
           primitive into the ReasonCode.Reason (ReasonTYPE)
           parameter value for the ED, AI or AD SPDU.

           NOTE: ReasonTYPE and ErActReasonTYPE are not
                 assignment compatible because they are
                 enumerated types.

 INPUTS:   Reason - the Reason parameter value of the
                    SUERreq, SACTIreq or SACTDreq primitive.

 OUTPUTS:  returns: the ReasonCode.Reason parameter value for the
                    ED, AI or AD SPDU.

 CALLS:    none.
}


PURE FUNCTION MapErActReq(Reason : ErActReasonTYPE) : ReasonTYPE;

BEGIN
  CASE Reason OF
    SSU_UNSPECIFIED : MapErActReq := SSU_UNSPECIFIED;
    SSU_CONGESTED   : MapErActReq := SSU_CONGESTED;
    SEQUENCE_ERROR  : MapErActReq := SEQUENCE_ERROR;
    LOCAL_SSU_ERROR : MapErActReq := LOCAL_SSU_ERROR;
    PROCEDURE_ERROR : MapErActReq := PROCEDURE_ERROR;
    DEMAND_DK       : MapErActReq := DEMAND_DK;
  END;
END;
```

```
{
 FUNCTION: MapErActInd

           This function maps the ReasonCode.Reason (ReasonTYPE)
           parameter value of the ED, AI or AD SPDU into the
           Reason (ErActReasonTYPE) parameter value for the
           SUERind, SACTIind or SACTDind primitive.

           NOTE: ReasonTYPE and ErActReasonTYPE are not
                 assignment compatible because they are
                 enumerated types.

 INPUTS:   Reason - the ReasonCode.Reason parameter value of the
                    ED, AI or AD SPDU.

 OUTPUTS:  returns: the Reason parameter value for the
                    SUERind, SACTIind or SACTDind primitive.

 CALLS:    none.
}


PURE FUNCTION MapErActInd(Reason : ReasonTYPE) : ErActReasonTYPE;

BEGIN
  CASE Reason OF
    SSU_UNSPECIFIED         : MapErActInd := SSU_UNSPECIFIED;
    SSU_CONGESTED           : MapErActInd := SSU_CONGESTED;
    SEQUENCE_ERROR          : MapErActInd := SEQUENCE_ERROR;
    LOCAL_SSU_ERROR         : MapErActInd := LOCAL_SSU_ERROR;
    PROCEDURE_ERROR         : MapErActInd := PROCEDURE_ERROR;
    DEMAND_DK               : MapErActInd := DEMAND_DK;

    SSU_SEE_DATA,
    CALLED_SSAP_UNKNOWN,
    CALLED_SSU_UNATTACHED,
    SSP_CONGESTED,
    PROPOSED_PROTOCOL       : ; {not for ED, AI or AD}
  END;
END;
```

```
{
 FUNCTION: MapRefRsp

           This function maps the Result (SCONrspResultTYPE)
           parameter value of the SCONrsp- primitive into the
           ReasonCode.Reason (ReasonTYPE) parameter value for the
           RF SPDU.

           NOTE: ReasonTYPE and SCONrspResultTYPE are not
                 assignment compatible because they are
                 enumerated types.

 INPUTS:   Result - the Result parameter value of the SCONrsp-
                    primitive.

 OUTPUTS:  returns: the ReasonCode.Reason parameter value for the
                    RF SPDU.

 CALLS:    none.
}


PURE FUNCTION MapRefRsp(Result : SCONrspResultTYPE) : ReasonTYPE;

BEGIN
  CASE Result OF
    ACCEPT           : ; {not for SCONrsp-}
    SSU_UNSPECIFIED : MapRefRsp := SSU_UNSPECIFIED;
    SSU_CONGESTED    : MapRefRsp := SSU_CONGESTED;
    SSU_SEE_DATA     : MapRefRsp := SSU_SEE_DATA;
  END;
END;
```

```
{
 FUNCTION: MapRefCnf

           This function maps the ReasonCode.Reason (ReasonTYPE)
           parameter value of the RF SPDU into the Result
           (SCONcnfResultTYPE) parameter value for the SCONcnf-
           primitive.

           NOTE: ReasonTYPE and SCONcnfResultTYPE are not
                 assignment compatible because they are
                 enumerated types.

 INPUTS:   Reason - the ReasonCode.Reason parameter value of the
                    RF SPDU.

 OUTPUTS:  returns: the Result parameter value for the SCONcnf-
                    primitive.

 CALLS:    none.
}


PURE FUNCTION MapRefCnf(Reason : ReasonTYPE) : SCONcnfResultTYPE;

BEGIN
  CASE Reason OF
    SSU_UNSPECIFIED       : MapRefCnf := SSU_UNSPECIFIED;
    SSU_CONGESTED         : MapRefCnf := SSU_CONGESTED;
    SSU_SEE_DATA          : MapRefCnf := SSU_SEE_DATA;
    CALLED_SSAP_UNKNOWN   : MapRefCnf := CALLED_SSAP_UNKNOWN;
    CALLED_SSU_UNATTACHED : MapRefCnf := CALLED_SSU_UNATTACED;
    SSP_CONGESTED         : MapRefCnf := SSP_CONGESTED;
    PROPOSED_PROTOCOL     : MapRefCnf := SSP_UNSPECIFIED;

    SEQUENCE_ERROR,
    LOCAL_SSU_ERROR,
    PROCEDURE_ERROR,
    DEMAND_DK             : ; {not for RF SPDU}
  END;
END;
```

```
{
 PROCEDURE: PIU building procedures

            Given a PIU parameter value and a TSDU, each of these
            procedures appends a unique PIU, as part of a SPDU,
            onto the TSDU. Since the inclusion of certain PIUs in
            a SPDU is non-mandatory, these procedures only append
            the PIU onto the TSDU if the relevant conditions are
            satisfied.

            These procedures only build those PIUs forming part
            of those SPDUs required by X.400.

 INPUTS:    tsdu      - the TSDU. Its .l field indicates its
                        current length.
            parameter - the PIU parameter value.

 OUTPUTS:   tsdu.l - is updated to include the appended PIU,
                     if any.

 CALLS:     BuildHeader, FU, AV.
}
```

```
{
 Build Called SS-user reference
}

PURE PROCEDURE Build9PIU(VAR tsdu          : TSDUTYPE;
                         CalledSSuserRef : Bytes64TYPE);

VAR PI    : ByteTYPE;
    LI,i : INTEGER;

BEGIN
  PI := 9;
  LI := CalledSSuserRef.l;

  IF LI > 0
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);

      FOR i := 1 TO LI DO
        tsdu.d[tsdu.l+i] := CalledSSuserRef.d[i];

      tsdu.l := tsdu.l + LI;
    END;
END;


{
 Build Calling SS-user reference
}

PURE PROCEDURE Build10PIU(VAR tsdu          : TSDUTYPE;
                          CallingSSuserRef : Bytes64TYPE);

VAR PI    : ByteTYPE;
    LI,i : INTEGER;

BEGIN
  PI := 10;
  LI := CallingSSuserRef.l;

  IF LI > 0
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);

      FOR i := 1 TO LI DO
        tsdu.d[tsdu.l+i] := CallingSSuserRef.d[i];

      tsdu.l := tsdu.l + LI;
    END;
END;
```

```
{
 Build Common reference
}

PURE PROCEDURE Build11PIU(VAR tsdu   : TSDUTYPE;
                          CommonRef : Bytes64TYPE);

VAR PI   : ByteTYPE;
    LI,i : INTEGER;

BEGIN
  PI := 11;
  LI := CommonRef.l;

  IF LI > 0
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);

      FOR i := 1 TO LI DO
        tsdu.d[tsdu.l+i] := CommonRef.d[i];

      tsdu.l := tsdu.l + LI;
    END;
END;


{
 Build Additional reference information
}

PURE PROCEDURE Build12PIU(VAR tsdu      : TSDUTYPE;
                          AdditionalRef : Bytes4TYPE);

VAR PI   : ByteTYPE;
    LI,i : INTEGER;

BEGIN
  PI := 12;
  LI := AdditionalRef.l;

  IF LI > 0
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);

      FOR i := 1 TO LI DO
        tsdu.d[tsdu.l+i] := AdditionalRef.d[i];

      tsdu.l := tsdu.l + LI;
    END;
END;
```

```
{
 Build Sync type item
}

PURE PROCEDURE Build15PIU(VAR tsdu     : TSDUTYPE;
                          SyncTypeItem : SyncTypeTYPE);

VAR PI : ByteTYPE;
    LI : INTEGER;

BEGIN
  PI := 15;
  LI := 1;

  IF SyncTypeItem = OPTIONAL
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);
      tsdu.d[tsdu.l+1] := 1;
      tsdu.l := tsdu.l + LI;
    END;
END;
```

```
{
 Build Token item
}

PURE PROCEDURE Build16PIU(VAR tsdu   : TSDUTYPE;
                          TokenItem : TokenSetTYPE);

VAR PI           : ByteTYPE;
    LI,byte,bit  : INTEGER;
    token        : TokenTYPE;

BEGIN
  PI := 16;
  LI := 1;

  IF TokenItem <> []
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);
      byte := 0;
      bit  := 1;

      FOR token := DKT TO TRT DO
        BEGIN
          IF token IN TokenItem
          THEN
            byte := byte + bit;
          bit := bit * 4;
        END;

      tsdu.d[tsdu.l+1] := byte;
      tsdu.l := tsdu.l + LI;
    END;
END;
```

```
{
 Build Transport disconnect
}

PURE PROCEDURE Buildl7PIU(VAR tsdu : TSDUTYPE;
                          TCdis    : TCdisTYPE);

VAR PI      : ByteTYPE;
    LI,byte : INTEGER;

BEGIN
  PI := 17;
  LI := 1;

  IF TRUE
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);

      CASE TCdis.TCkept OF
         TRUE : byte := 0;
        FALSE : byte := 1;
      END;

      CASE TCdis.ABreason OF
        NO_ABORT       : byte := byte + 0;
        USER_ABORT     : byte := byte + 2;
        PROTOCOL_ERROR : byte := byte + 4;
        NO_REASON      : byte := byte + 8;
      END;

      tsdu.d[tsdu.l+1] := byte;
      tsdu.l := tsdu.l + LI;
    END;
END;
```

```
{
 Build Protocol options
}

PURE PROCEDURE Build19PIU(VAR tsdu          : TSDUTYPE;
                          ProtocolOptions : ByteTYPE);

VAR PI : ByteTYPE;
    LI : INTEGER;

BEGIN
  PI := 19;
  LI := 1;

  IF TRUE
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);
      tsdu.d[tsdu.l+1] := ProtocolOptions;
      tsdu.l := tsdu.l + LI;
    END;
END;
```

```
{
 Build Session user requirements
}

PURE PROCEDURE Build20PIU(VAR tsdu       : TSDUTYPE;
                          Srequirements : FUsetTYPE);

VAR PI                  : ByteTYPE;
    LI,total,bitweight : INTEGER;
    fu                  : FUTYPE;

BEGIN
  PI := 20;
  LI := 2;

  IF TRUE
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);
      total := 0;
      bitweight := 1;

      FOR fu := HD TO TD DO
        BEGIN
          IF fu IN Srequirements
          THEN
            total := total + bitweight;
          bitweight := bitweight * 2;
        END;

      tsdu.d[tsdu.l+1] := total DIV 256;
      tsdu.d[tsdu.l+2] := total MOD 256;
      tsdu.l := tsdu.l + LI;
END;
```

```
{
 Build TSDU maximum size
}

PURE PROCEDURE Build21PIU(VAR tsdu     : TSDUTYPE;
                          maxTSDUlen0 : INTEGER;
                          maxTSDUlen1 : INTEGER);

VAR PI : ByteTYPE;
    LI : INTEGER;

BEGIN
  PI := 21;
  LI := 4;

  IF (maxTSDUlen0 > 0) OR (maxTSDUlen1 > 0)
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);
      tsdu.d[tsdu.l+1] := maxTSDUlen0 DIV 256;
      tsdu.d[tsdu.l+2] := maxTSDUlen0 MOD 256;
      tsdu.d[tsdu.l+3] := maxTSDUlen1 DIV 256;
      tsdu.d[tsdu.l+4] := maxTSDUlen1 MOD 256;
      tsdu.l := tsdu.l + LI;
    END;
END;


{
 Build Version number
}

PURE PROCEDURE Build22PIU(VAR tsdu       : TSDUTYPE;
                          VersionNumber : ByteTYPE);

VAR PI : ByteTYPE;
    LI : INTEGER;

BEGIN
  PI := 22;
  LI := 1;

  IF TRUE
  THEN
    BuildHeader(tsdu,PI,LI);
    tsdu.d[tsdu.l+1] := VersionNumber;
    tsdu.l := tsdu.l + LI;
  END;
END;
```

b) Intersection between STA713 (data transfer) and SUERind: Predicate p50 (FU(EXCEP) & (^FU(ACT) OR Vact) & AA(dk)) will never be true because the RTS must own the tokens to receive SUERind.

From TABLE A-14/X.215, Connection release state table:

a) Intersection between STA09 (await SRELrsp) and SRELcnf+: Intersection between STA03 (await SRELcnf) and SRELind: These intersections result from a collision of release requests, which can never happen to the RTS because only the sending RTS may request session connection release.

b) Intersection between STA09 (await SRELrsp) and SRELrsp+: Predicate p69 (Vcoll) will only be true if release requests collide, which can never happen to the RTS because only the sending RTS may request session connection release.

7) Of the remaining state table intersections, some have conditional action lists for which the boolean predicates (as defined in CCITT Recommendation X.215 TABLE A-6/X.215) will always be true for RTS use. These predicates have therefore no effect on RTS operation and may be omitted. These predicates are:

a) ^p69 (^Vcoll) may be omitted because Vcoll, although not used by the RTS, will always be false for RTS use. Vcoll only becomes true when release requests collide, which never happens to RTSs, as previously explained.

8) Of the remaining state table intersections, some specify actions which have no effect on the RTS. These may therefore be omitted from the action lists.

Specific actions (as defined in CCITT Recommendation X.215 TABLE A-5/X.215) which occur in the remaining intersections but which may be omitted are:

a) From [5] the actions "Set V(R) = 0", "Set Vcoll = false" and "Set Vrsp = no" may be omitted because V(R), Vcoll and Vrsp are not used by the RTS.

b) From [22] the action "Set V(R) = V(M)" may be omitted because V(R) is not used by the RTS.

c) From [26] the action "Set V(R) = 1" may be omitted because V(R) is not used by the RTS.

d) From [27] the action "Set V(R) = 1" may be omitted because V(R) is not used by the RTS.

The resultant state tables describing the session service for the RTS are presented below:

Table A.1 Connection establishment state table

INCOMING EVENT	CURRENT STATE		
	STA01 idle, no SC	STA02A await SCONcnf	STA08 await SCONrsp
SCONcnf+		[5][11] STA713	
SCONcnf-		STA01	
SCONind	STA08		
SCONreq	STA02A		
SCONrsp+			[5][11] STA713
SCONrsp-			STA01

Table A.2 Data transfer state table

INCOMING EVENT	CURRENT STATE
	STA713 data transfer
SDTind	STA713
SDTreq	p03 STA713

Table A.3 Synchronization state table

INCOMING EVENT	CURRENT STATE				
	STA03 await SRELcnf	STA04B await SACTEcnf	STA09 await SRELrsp	STA10B await SACTErsp	STA713 data transfer
SACTEcnf		[14][22] STA713			
SACTEind					[23] STA10B
SACTEreq					p71 [24] STA04B
SACTErsp				[14][22] STA713	
SSYNmcnf	[25] STA03	[25] STA04B			[25] STA713
SSYNmind					[23] STA713
SSYNmreq					p15 [24] STA713
SSYNmrsp			p18&p21 [25] STA09	p18&^p20 &p21 [25] STA10B	p18&p21 [25] STA713

Table A.4 (part 1 of 2) Activity interrupt and discard state table

INCOMING EVENT	CURRENT STATE				
	STA04B await SACTEcnf	STA05B await SACTIcnf	STA05C await SACTDcnf	STA10B await SACTErsp	STA11B await SACTIrsp
SACTDcnf			[29] STA713		
SACTDind				STA11C	
SACTDreq	p39 STA05C				
SACTDrsp					
SACTIcnf		[29] STA713			
SACTIind				STA11B	
SACTIreq	p39 STA05B				
SACTIrsp					[30] STA713

Table A.4 (part 2 of 2) Activity interrupt and discard state table

INCOMING EVENT	CURRENT STATE			
	STA11C await SACTDrsp	STA19 await recovery indicate	STA20 await recovery request	STA713 data transfer
SACTDcnf				
SACTDind		STA11C		STA11C
SACTDreq			p34&p11 STA05C	p34&p39 STA05C
SACTDrsp	[30] STA713			
SACTIcnf				
SACTIind		STA11B		STA11B
SACTIreq			p34&p11 STA05B	p34&p39 STA05B
SACTIrsp				

Table A.5 Activity start and resume state table

INCOMING EVENT	CURRENT STATE
	STA713 data transfer
SACTRind	[12][27] STA713
SACTRreq	p45 [12][27] STA713
SACTSind	[12][26] STA713
SACTSreq	p45 [12][26] STA713

Table A.6 (part 1 of 2)

Token management and exceptions state table

INCOMING EVENT	CURRENT STATE				
	STA03 await SRELcnf	STA04B await SACTEcnf	STA09 await SRELrsp	STA10B await SACTErsp	STA19 await recovery indicate
SCGind					
SCGreq					
SPTind	STA03	STA04B			
SPTreq			p53 STA09	p53 STA10B	
SUERind	STA20	STA20			
SUERreq			p50 STA19	p50 STA19	

Table A.6 (part 2 of 2)

Token management and exceptions state table

INCOMING EVENT	CURRENT STATE
	STA713 data transfer
SCGind	[11] STA713
SCGreq	p55 [11] STA713
SPTind	STA713
SPTreq	p53 STA713
SUERind	p51 STA20
SUERreq	p50 STA19

Table A.7 Connection release state table

INCOMING EVENT	CURRENT STATE			
	STA03 await SRELcnf	STA09 await SRELrsp	STA713 data transfer	any other state
SPABind	STA01	STA01	STA01	STA01
SRELcnf+	STA01			
SRELind			STA09	
SRELreq			p63 STA03	
SRELrsp+		STA01		
SUABind	STA01	STA01	STA01	STA01
SUABreq	STA01	STA01	STA01	STA01

APPENDIX B. Session Protocol State Tables for the X.400 SPM

This appendix describes the session protocol as performed by the X.400 SPM in terms of state tables. It shows how these are derived from those of CCITT Recommendation X.225 ANNEX A, which describes the general session protocol in terms of state tables. The state tables of this appendix use the notation, conventions and definitions established in CCITT Recommendation X.225 ANNEX A. These issues will therefore not be repeated here and a thorough knowledge of them will be assumed.

The X.400 SPM state tables are derived from the general state tables by extracting from the latter only those elements used by the X.400 SPM. This is achieved by simply omitting all those elements of the general state tables which are not used by the X.400 SPM. This process consists of the following nine sequential steps:

1) CCITT Recommendation X.225 ANNEX A.4.1.2 and A.4.2.2 specify two possible courses of action to be taken by the SPM on detection of either a protocol error or a conditional action list for which none of the predicate expressions are true. The X.400 SPM cannot take course b) because this assumes that the SPM provides the Provider Exception Reporting service, which is not provided by the X.400 SPM. The X.400 SPM must therefore take course a), which specifies that the SPM shall:

    1) issue a S-P-ABORT.indication;
    2) send an ABORT SPDU;
    3) start the timer, TIM;
    4) enter STA16 and wait for a T-DISCONNECT.indication or an ABORT ACCEPT SPDU.

2) CCITT Recommendation X.225 ANNEX A.4.3 specifies four possible courses of action to be taken by the SPM on receipt of an invalid SPDU. Before one of these courses are selected, it must be noted that the X.400 SPM described in this thesis does not perform any validation of incoming SPDU structure and/or encoding. The reason for this is based on the following two, reasonable assumptions:

a) the remote SPM does not make errors in constructing and encoding SPDUs; and
b) the Transport Layer provides an error-free transfer of SPDUs between correspondent SPMs.

Of the four courses of action, courses a), b) and c) are not taken by the X.400 SPM because it cannot detect invalid SPDUs. This leaves course d) - "take no action" - as the most appropriate to be taken by the X.400 SPM.

3) Certain of the sets and variables defined in CCITT Recommendation X.225 ANNEX A.5 are not required by the X.400 SPM and may therefore be omitted. These are:

a) From A.5.3:
The subset of tokens **GT = {tokens given in the input event}** may be omitted because it is used only by the Give Tokens service, which is not provided by the X.400 SPM.

b) From A.5.4.4:
The variables **Vrsp** and **Vrspnb** may be omitted because they are used only by the Resynchronization service, which is not provided by the X.400 SPM.

c) From A.5.4.5:
The function **SPMwinner** may be omitted because it is used only by the Resynchronization service, which is not provided by the X.400 SPM.

d) From A.5.4.8:

The variable **Vcoll** may be omitted because it is used only when FINISH SPDUs collide. This can never happen to X.400 SPMs because only the X.400 SPM which owns the tokens may send the FINISH SPDU.

e) From A.5.4.11:

The variable **V(R)** may be omitted because it is used only by the Resynchronization service, which is not provided by the X.400 SPM.

4) All state table rows representing incoming SS-user events (as defined in CCITT Recommendation X.225 TABLE A-1/X.225) associated with those services not provided by the X.400 SPM are omitted. These events are:

abbreviated name	name and description
SCDreq	S-CAPABILITY-DATA.request
SCDrsp	S-CAPABILITY-DATA.response
SEXreq	S-EXPEDITED-DATA.request
SGTreq	S-TOKEN-GIVE.request
SRELrsp-	S-RELEASE.response (reject)
SRSYNreq	S-RESYNCHRONIZE.request
SRSYNrsp	S-RESYNCHRONIZE.response
SSYNMreq	S-SYNC-MAJOR.request
SSYNMrsp	S-SYNC-MAJOR.response
STDreq	S-TYPED-DATA.request

5) All state table rows representing incoming SPDU events (as defined in CCITT Recommendation X.225 TABLE A-1/X.225) for those SPDUs not used by the X.400 SPM are omitted. These SPDUs are:

SPDU code	SPDU name
CD	CAPABILITY DATA
CDA	CAPABILITY DATA ACK
ER	EXCEPTION REPORT
EX	EXPEDITED DATA
GT	GIVE TOKENS with Token Item parameter (Note 1)
MAA	MAJOR SYNC ACK
MAP	MAJOR SYNC POINT
NF	NOT FINISHED
PR-MAA	PREPARE (MAJOR SYNC ACK)
PR-RA	PREPARE (RESYNCHRONIZE ACK)
PR-RS	PREPARE (RESYNCHRONIZE)
RA	RESYNCHRONIZE ACK
RS-a	RESYNCHRONIZE (abandon)
RS-r	RESYNCHRONIZE (restart)
RS-s	RESYNCHRONIZE (set)
TD	TYPED DATA

Note 1:
A GT SPDU (or a PT SPDU) without the Token Item parameter is used to introduce a concatenated sequence of SPDUs, Basic Concatenation in the case of the X.400 SPM. Concatenation and separation of SPDUs are not handled by the state tables.

6) All state table columns representing states (as defined in CCITT Recommendation X.225 TABLE A-2/X.225) associated with those SS-user events and SPDUs not used by the X.400 SPM are omitted. These states are:

abbreviated name	name and description
STA04A	await MAA or PR-MAA SPDU
STA05A	await RA or PR-RA SPDU
STA06	await RS SPDU
STA10A	await SSYNMrsp
STA11A	await SRSYNrsp
STA15A	after PR SPDU, await MAA or AEA SPDU
STA15B	after PR SPDU, await RS, AI or AD SPDU
STA15C	after PR SPDU, await RA, AIA or ADA SPDU
STA21	await CDA SPDU
STA22	await SCDrsp

7) Of the remaining state table intersections, some have conditional action lists for which the boolean predicate conditions (as defined in CCITT Recommendation X.225 TABLE A-6/X.225) will never be true for X.400 SPM use. These actions lists will therefore never be performed by the X.400 SPM and may therefore be omitted. These intersections are:

From TABLE A-8/X.225, Data transfer state table:

a) Intersection between STA03 (await DN) and DT:
Intersection between STA04B (await AEA) and DT:
Intersection between STA05B (await AIA) and DT:
Intersection between STA05C (await ADA) and DT:
Intersection between STA20 (await recovery) and DT:
Predicate p05 (A(dk)) will never be true because to enter any of these states the X.400 SPM must own the tokens.

b) Intersection between STA18 (await GTA) and DT:
Intersection between STA18 (await GTA) and SDTreq:
Predicate p70 (FU(FD)) will never be true because the X.400 SPM does not provide the FD functional unit.

c) Intersection between STA09 (await SRELrsp) and SDTreq:
Predicate p04 (FU(FD) & ^Vcoll) will never be true because the X.400 SPM does not provide the FD functional unit.

d) Intersection between STA10B (await SACTErsp) and SDTreq:
Predicate p03 (I(dk)) will never be true because to enter STA10B the X.400 SPM must not own the tokens.

From TABLE A-9/X.225, Synchronization state table:

a) Intersection between STA20 (await recovery) and AE:
Predicate p72 (FU(ACT) & Vact & A(dk) & A(mi) & AA(ma)) will never be true because to enter STA20 the X.400 SPM must own the tokens.

b) Intersection between STA19 (await recovery (init)) and MIA:
For the X.400 SPM to enter STA19 it must not own the tokens, while only the owner of the tokens may receive MIA.

c) Intersection between STA20 (await recovery) and MIP:
Predicate p14 ((^FU(ACT) OR Vact) & A(dk) & AA(mi)) will never be true because to enter STA20 the X.400 SPM must own the tokens.

From TABLE A-11/X.225, Activity interrupt and discard state table:

a) Intersection between STA20 (await recovery) and AD:
Intersection between STA20 (await recovery) and AI:
Predicate p40 (AA(ma)) will never be true because to enter STA20 the X.400 SPM must own the tokens.

From TABLE A-13/X.225, Token management and exceptions state table:

a) Intersection between STA19 (await recovery (init)) and ED:
Predicate p51 (FU(EXCEP) & (^FU(ACT) OR Vact) & II(dk)) will never be true because to enter STA19 the X.400 SPM must not own the tokens.

b) Intersection between STA713 (data transfer) and ED:
Predicate p50 (FU(EXCEP) & (^FU(ACT) OR Vact) & AA(dk)) will never be true because the X.400 SPM may only receive ED when it owns the tokens.

From TABLE A-14/X.225, Connection release state table:

a) Intersection between STA09 (await SRELrsp) and DN:
Intersection between STA09 (await SRELrsp) and SRELrsp+:
Predicate p69 (Vcoll) will never be true because FINISH SPDUs from X.400 SPMs cannot collide. Only the X.400 SPM which owns the tokens may send the FINISH SPDU.

b) Intersection between STA09 (await SRELrsp) and SRELreq:
Intersection between STA03 (await DN) and FNnr:
Intersection between STA03 (await DN) and FNr:
Predicate ^p65 (^ANY(AV,tk_dom)) will never be true because there are always tokens available to the X.400 SPM.

8) Of the remaining state table intersections, some have conditional action lists for which the boolean predicates (as defined in CCITT Recommendation X.225 TABLE A-6/X.225) will always be true for X.400 SPM use. These predicates have therefore no effect on X.400 SPM operation and may be omitted. These predicates are:

a) ^p69 (^Vcoll) may be omitted because Vcoll, although not used by the X.400 SPM, will always be false for X.400 SPM use. Vcoll only becomes true when FINISH SPDUs collide, which never happens to X.400 SPMs, as previously explained.

9) Of the remaining state table intersections, some specify actions which have no effect on the X.400 SPM. These may therefore be omitted from the action lists.

Specific actions (as defined in CCITT Recommendation X.225 TABLE A-5/X.225) which occur in the remaining intersections but which may be omitted are:

a) From [5] the actions "Set V(R) = 0", "Set Vcoll = false" and "Set Vrsp = no" may be omitted because V(R), Vcoll and Vrsp are not used by the X.400 SPM.

b) [6] may be omitted because it performs operations on an event queue which is only used by the Expedited Data Transfer service, which is not provided by the X.400 SPM.

c) [16] may be omitted because it concerns the variables Vrsp and Vrspnb, which are not used by the X.400 SPM.

d) From [22] the action "Set V(R) = V(M)" may be omitted because V(R) is not used by the X.400 SPM.

e) From [26] the action "Set V(R) = 1" may be omitted because V(R) is not used by the X.400 SPM.

f) From [27] the action "Set V(R) = 1" may be omitted because V(R) is not used by the X.400 SPM.

g) From [29] the action "set Vrsp = no" may be omitted because Vrsp is not used by the X.400 SPM.

h) From [30] the action "set Vrsp = no" may be omitted because Vrsp is not used by the X.400 SPM.

Of the remaining state table intersections, some action lists specify that the PREPARE SPDU is to be sent if TEXP is true (i.e., if the Transport Expedited Data Transfer service option has been selected for this session connection). Since TEXP will always be false for the X.400 SPM, PREPARE will never be sent and this action may therefore be omitted.

The resultant state tables describing the session protocol performed by the X.400 SPM are presented below:

Table B.1 (part 1 of 2)

Connection establishment state table

INCOMING EVENT	CURRENT STATE			
	STA01 idle, no TC	STA01A await AA	STA01B await TCONcnf	STA01C idle, TC con
AC	//	STA01A	//	TDISreq STA01
CN	//	TDISreq [3] STA01	//	^p01 SCONind STA08 p01 TDISreq STA01
RFnr	//	STA01A	//	TDISreq STA01
RFr	//	STA01A	//	TDISreq STA01
SCONreq	TCONreq [2] STA01B			p01 CN STA02A
SCONrsp+				
SCONrsp-				
TCONcnf	//	//	CN STA02A	//
TCONind	TCONrsp [1] STA01C	//	//	//

Table B.1 (part 2 of 2)

Connection establishment state table

INCOMING EVENT	CURRENT STATE		
	STA02A await AC	STA08 await SCONrsp	STA16 await TDISind
AC	SCONcnf+ [5] [11] STA713		STA16
CN			TDISreq [3] STA01
RFnr	SCONcnf- TDISreq STA01		STA16
RFr	^p02 SCONcnf- TDISreq STA01 p02 SCONcnf- STA01C		STA16
SCONreq			
SCONrsp+		AC [5] [11] STA713	
SCONrsp-		^p02 RFnr [4] STA16 p02 RFr STA01C	
TCONcnf	//	//	//
TCONind	//	//	//

Table B.2

Data transfer state table

INCOMING EVENT	CURRENT STATE				
	STA01A await AA	STA01C idle, TC con	STA16 await TDISind	STA19 await recovery (init)	STA713 data transfer
DT	STA01A	TDISreq STA01	STA16	STA19	p05 SDTind STA713
SDTreq					p03 DT STA713

Table B.3 (part 1 of 3)

Synchronization state table

INCOMING EVENT	CURRENT STATE			
	STA01A await AA	STA01C idle, TC con	STA03 await DN	STA04B await AEA
AEA	STA01A	TDISreq STA01		p16&p20 SACTEcnf [14][22] STA713
AE	STA01A	TDISreq STA01		
MIA	STA01A	TDISreq STA01	p17&p21 SSYNmcnf [25] STA03	p17&^p20 &p21 SSYNmcnf [25] STA04B
MIP	STA01A	TDISreq STA01		
SACTEreq				
SACTErsp				
SSYNmreq				
SSYNmrsp				

Table B.3 (part 2 of 3)

Synchronization state table

INCOMING EVENT	CURRENT STATE			
	STA05B await AIA	STA05C await ADA	STA09 await SRELrsp	STA10B await SACTErsp
AEA	STA05B	STA05C		
AE				
MIA	p17 STA05B	p17 STA05C		
MIP				
SACTEreq				
SACTErsp				AEA [14][22] STA713
SSYNmreq				
SSYNmrsp			p18&p21 MIA [25] STA09	p18&^p20 &p21 MIA [25] STA10B

Table B.3 (part 3 of 3)

Synchronization state table

INCOMING EVENT	CURRENT STATE			
	STA16 await TDISind	STA19 await recovery (init)	STA20 await recovery	STA713 data transfer
AEA	STA16		p20 STA20	
AE	STA16	p72&p19 [31] STA19		p72&p19 SACTEind [13][31] STA10B
MIA	STA16		p17&p21 STA20	p17&p21 SSYNmcnf [25] STA713
MIP	STA16	p14&p19 [23] STA19		p14&p19 SSYNmind [23] STA713
SACTEreq				p71 AE [13][24] STA04B
SACTErsp				
SSYNmreq				p15 MIP [24] STA713
SSYNmrsp				p18&p21 MIA [25] STA713

Table B.4 (part 1 of 3)

Activity interrupt and discard state table

INCOMING EVENT	CURRENT STATE			
	STA01A await AA	STA01C idle, TC con	STA04B await AEA	STA05B await AIA
AD	STA01A	TDISreq STA01		
ADA	STA01A	TDISreq STA01		
AI	STA01A	TDISreq STA01		
AIA	STA01A	TDISreq STA01		p38 SACTIcnf [29] STA713
SACTDreq			p39 AD STA05C	
SACTDrsp				
SACTIreq			p39 AI STA05B	
SACTIrsp				

Table B.4 (part 2 of 3)

Activity interrupt and discard state table

INCOMING EVENT	CURRENT STATE			
	STA05C await ADA	STA10B await SACTErsp	STA11B await SACTIrsp	STA11C await SACTDrsp
AD		p38&p40 SACTDind STA11C		
ADA	p38 SACTDcnf [29] STA713			
AI		p38&p40 SACTIind STA11B		
AIA				
SACTDreq				
SACTDrsp				ADA [30] STA713
SACTIreq				
SACTIrsp			AIA [30] STA713	

Table B.4 (part 3 of 3)

Activity interrupt and discard state table

INCOMING EVENT	CURRENT STATE			
	STA16 await TDISind	STA19 await recovery (init)	STA20 await recovery	STA713 data transfer
AD	STA16	p38&p40 SACTDind STA11C		p38&p40 SACTDind STA11C
ADA	STA16			
AI	STA16	p38&p40 SACTIind STA11B		p38&p40 SACTIind STA11B
AIA	STA16			
SACTDreq			p34&p11 AD STA05C	p34&p39 AD STA05C
SACTDrsp				
SACTIreq			p34&p11 AI STA05B	p34&p39 AI STA05B
SACTIrsp				

Table B.5

Activity start and resume state table

INCOMING EVENT	CURRENT STATE			
	STA01A await AA	STA01C idle, TC con	STA16 await TDISind	STA713 data transfer
AR	STA01A	TDISreq STA01	STA16	p44 SACTRind [12][27] STA713
AS	STA01A	TDISreq STA01	STA16	p44 SACTSind [12][26] STA713
SACTRreq				p45 AR [12][27] STA713
SACTSreq				p45 AS [12][26] STA713

Table B.6 (part 2 of 3)

Token management and exceptions state table

INCOMING EVENT	CURRENT STATE			
	STA05B await AIA	STA05C await ADA	STA09 await SRELrsp	STA10B await SACTErsp
ED	p48 STA05B	p48 STA05C		
GTA				
GTC				
PT	p53 STA05B	p53 STA05C		
SCGreq				
SPTreq			p53 PT STA09	p53 PT STA10B
SUERreq			p50 ED STA19	p50 ED STA19

A SESSION LAYER FOR THE X.400 MESSAGE HANDLING SYSTEM

by

EUGENE DANIEL VAN DER WESTHUIZEN

Submitted in fulfilment of the requirements for

the degree of

Master of Science

in the

Department of Electrical and Electronic Engineering

University of Cape Town

February 1990

Preface

The work for, and preparation of, this thesis were done while the author was a full time student in the Department of Electrical and Electronic Engineering at the University of Cape Town from March 1987 to January 1990. Supervision was by Mr. M.J.E. Ventura.

These studies represent original work by the author and have not been submitted in any other form to another university. Where use was made of the work of others it has been duly acknowledged in the text.

Signed: .....

E.D. van der Westhuizen

Department of Electrical and Electronic Engineering
University of Cape Town

Acknowledgments

The author would like to thank the following persons and institutions:

Mr. M.J.E. Ventura of the Department of Electrical and Electronic Engineering, my supervisor, for his help and guidance with this thesis, and for reading the final script.

Mr. Graham Jack of the Department of Electrical and Electronic Engineering for invaluable, practical help with any conceivable computer-related issues.

The South African Council for Scientific and Industrial Research for financial support for this thesis in the form of a post graduate bursary.

Abstract

The CCITT X.400 Message Handling System resides in the Application Layer of the seven-layer Reference Model for Open Systems Interconnection. It bypasses the services of the Presentation Layer completely to interact directly with the Session Layer.

The objectives of this thesis are to show how the general Session Layer may be tailored to be minimally conformant to the requirements of X.400; to produce a formal specification of this session layer; and to show how this session layer may be implemented on a real system.

The session services required by X.400 are those of the Half-duplex, Minor Synchronization, Exceptions and Activity Management functional units of the CCITT X.215 Session Service Definition. These services, and particularly their use by X.400, are described in detail. State tables describing these services are derived from the general session service state tables.

Those elements of the CCITT X.225 Session Protocol Specification which are required to provide only those services required by X.400 are described in detail. State tables describing this session protocol are derived from the general session protocol state tables.

A formal specification of the session layer for X.400 is presented using the Formal Description Technique Estelle. This specification includes a complete session entity, which characterizes the entire session layer for X.400.

A session entity for supporting X.400 is partially implemented and interfaced to an existing X.400 product on a real system. Only the Session Connection Establishment Phase of the session protocol is implemented to illustrate the technique whereby the entire session protocol may be implemented. This implementation uses the C programming language in the UNIX operating system environment.

TABLE OF CONTENTS

List of figures

List of tables

List of acronyms

APDU - Application Protocol Data Unit
CCITT - The International Telegraph and Telephone Consultative Committee
CEP - Connection End Point
ECMA - European Computer Manufacturers' Association
FDT - Formal Description Technique
FIFO - First In First Out
ISO - The International Standards Organization
MH - Message Handling
MHS - Message Handling System
MTA - Message Transfer Agent
OSI - Open Systems Interconnection
QOSS - Quality Of Session Service
QOTS - Quality Of Transport Service
RTS - Reliable Transfer Server
SAP - Service Access Point
SC - Session Connection
SCEP - Session Connection End Point
SIDU - Session Interface Data Unit
SPDU - Session Protocol Data Unit
SPM - Session Protocol Machine
SS - Session Service
SSAP - Session Service Access Point
SSDU - Session Service Data Unit
TC - Transport Connection
TCEP - Transport Connection End Point
TIDU - Transport Interface Data Unit
TPDU - Transport Protocol Data Unit
TS - Transport Service
TSAP - Transport Service Access Point
TSDU - Transport Service Data Unit
UA - User Agent

1. INTRODUCTION

The CCITT has recently (1985) defined a new, generic, global telecommunications service known as Message Handling. This service combines computer-based electronic mail systems with established telematic services such as telex and facsimile. This service is defined in the CCITT X.400 series of Message Handling System (MHS) Recommendations [1]. The entities providing this service reside in the application layer (layer seven) of the Reference Model of Open Systems Interconnection (OSI) [2] and make extensive use of the lower layers of the Model.

Of particular interest to X.400 is the session layer, the fifth layer of the OSI Reference Model. This layer provides dialogue control and management services to application entities through layer six, the presentation layer. However, the X.400 lower-layer service requirements have been designed to by-pass the presentation layer altogether, so that the X.400 application entities interact directly with the session layer. The general session layer used by X.400 is defined by the Session Service Definition of CCITT Recommendation X.215 [3] and the Session Protocol Specification of CCITT Recommendation X.225 [4].

The general session layer is a large and complex layer providing many optional services to meet the requirements of any type of application. X.400, however, uses only a subset of all possible session services, and may therefore be supported by a tailored version of the general session layer which provides only those services it needs. Describing and implementing such a session layer for X.400 is the subject of this thesis.

The objectives of this thesis are:

1. to show how the general session layer may be tailored to meet only the requirements of X.400;

2. to present a formal description of this session layer;

3. to show how this session layer may be implemented and interfaced to an existing X.400 application on a real system.

The reader is assumed to be thoroughly familiar with the concepts and terminology of the OSI Reference Model [2] and the OSI Layer Service Definition Conventions [5]. In addition, a thorough knowledge of the ISO Formal Description Technique *Estelle* [6], the C Programming Language [7] and the UNIX Operating System [8] is assumed when the formal description and implementation of the session layer is presented.

The material presented in the rest of this thesis is organized as follows:

Section 2 presents an overview of the general session layer. It briefly reviews all relevant concepts and terminology of the OSI Reference Model and OSI Layer Service Definition Conventions. It then broadly describes the purpose of the session layer, its use and provision of layer services, and elements of its internal operation.

Section 3 presents an overview of the X.400 Message Handling System. It briefly describes its architecture and shows why the session layer is of special importance to it.

Section 4 presents a detailed definition of the session service which is minimally conformant to the requirements of X.400 only. This is derived from the general Session Service Definition of CCITT Recommendation X.215.

This information is required by section 5, which presents a detailed specification of the session protocol required to provide only those session services required by X.400. This is derived from the general Session Protocol Specification of CCITT Recommendation X.225.

These informal descriptions of the session service and protocol required by X.400 are combined by section 6 into a complete, formal description of the session layer for X.400, using the ISO Formal Description Technique, Estelle.

Based on this formal description, section 7 shows how this session layer may be implemented on a real system. It presents a partial implementation of this session layer, written in the C programming language and interfaced to an existing X.400 application in the UNIX operating system environment.

Finally, section 8 concludes whether this description and implementation of the session layer for X.400 meets the objectives of this thesis.

2. OVERVIEW OF THE SESSION LAYER

This section presents a general overview of the session layer. First, it shows how the session layer fits into the OSI environment, identifying and reviewing relevant concepts and terminology of the OSI Reference Model and the OSI Layer Service Definition Conventions. These concepts are not thoroughly defined here, as such definitions may be found in CCITT Recommendations X.200 [2] and X.210 [5]. This is followed by a description of the purpose of the session layer. Broad descriptions are then given of the services available to the session layer from the transport layer, the services provided by the session layer to its users, and the major, internal session layer functions.

2.1 The session layer in the OSI environment

Figure 2.1 depicts the session layer in the OSI environment, showing elements of its internal structure and interaction with adjacent layers. This is followed by a brief description of each element.

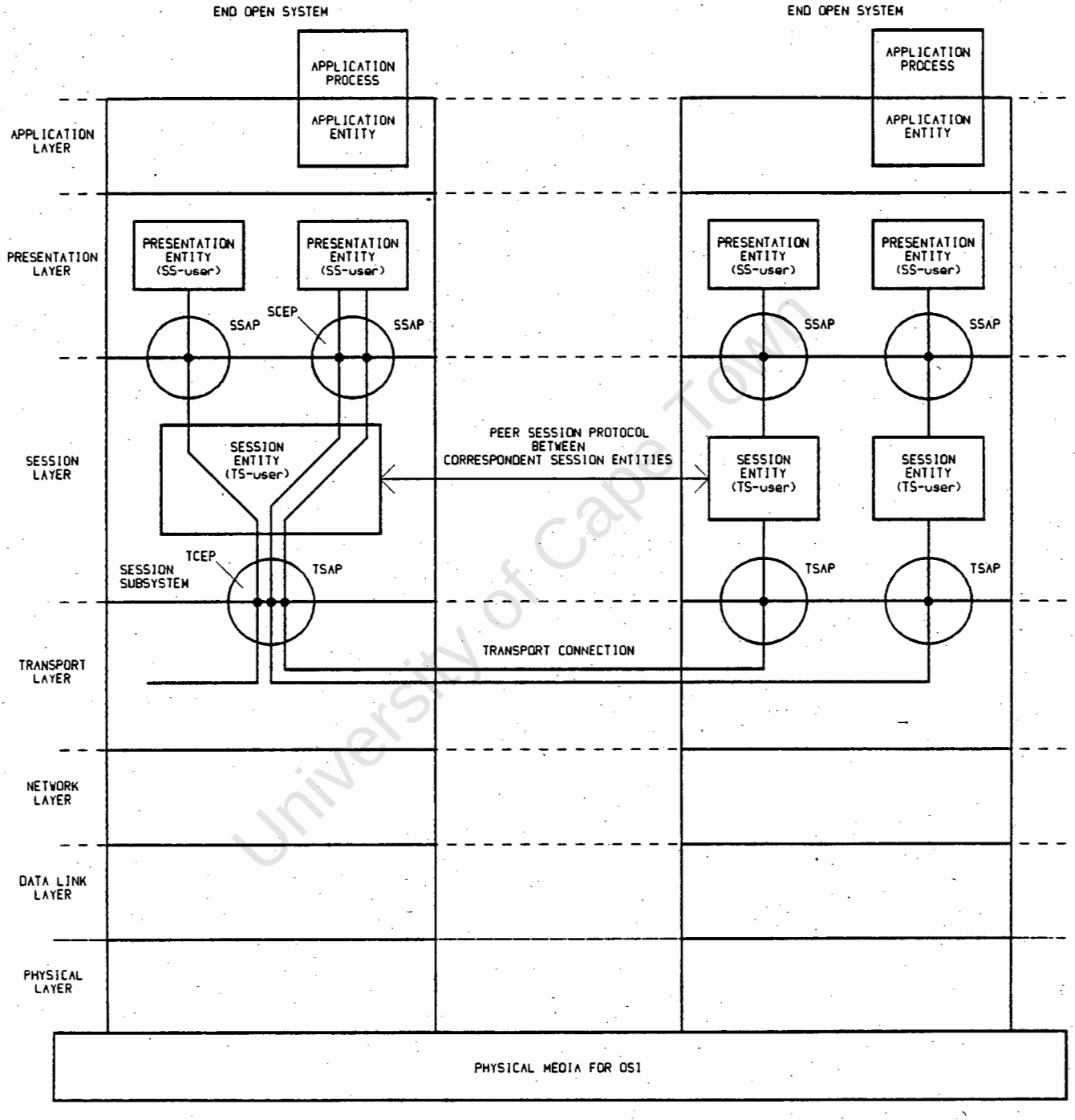

Figure 2.1 The session layer in the OSI environment

Layering:

Open systems are real systems which employ the standardized communication procedures derived from the OSI Reference Model.

An *application process* performs information processing for an application, while an *application entity* represents those aspects of the application process of concern to OSI.

A *session subsystem* is that element of the hierarchical division of an open system representing session layer functionality. The session subsystems in all open systems collectively form the *session layer*. A session subsystem consists of one or more active elements called *session entities*. All session entities within the session layer are called *peer* session entities.

A *service* is a capability provided by one layer to the layer above it. Session entities are responsible for providing *session services* directly to presentation entities. A *Session Service Access Point* (SSAP) is the point at which one session entity provides session services to one presentation entity. In providing the session service, session entities communicate with each other using the (peer) *session protocol* via services provided by the transport layer.

A *Transport Service Access Point* (TSAP) is the point at which one session entity uses transport services provided by one transport entity.

Communication between peer entities:

A *session connection* is an association for communication established by the session layer between two correspondent presentation entities. Session connections are provided between two SSAPs, where they are terminated by *Session Connection Endpoints* (SCEPs). Similarly, correspondent session entities communicate via a *transport connection* accessible at *Transport Connection End Points* (TCEPs) within their TSAPs.

Identifiers:

A presentation entity is uniquely identified by its SSAP *address*, or *session address*. Similarly, a session entity is uniquely identified by its TSAP *address*, or *transport address*. Connection endpoints within a SAP are identified by *Connection Endpoint identifiers*, which are unique within the scope of the entity supported by the SAP.

Addressing:

In the application layer, applications are referenced by a *directory* function which maps *application titles* into the PSAP addresses through which they may be accessed. Below the transport layer, a directory function provides the mapping between an NSAP address and the routing information required to create a path to the destination NSAP.

For the middle layers (presentation, session and transport), a unique (N)-address consists of its unique, supporting (N-1)-address plus an *(N)-suffix* which is unique within the scope of the (N-1)-address. This leads to a simple *hierarchical* address arrangement. An (N)-suffix is also called an *(N)-SAP selector* or an *(N)-SAP identifier*. The use of selectors is not mandatory, allowing *one-to-one mappings* between (N) and (N-1) addresses. The address information for a given layer (the (N)-

suffix, selector or identifier) is always conveyed within the protocol of that layer.

Using this addressing scheme, the address of an application entity can therefore only ever comprise:

Application address = NSAP address + TSAP selector
+ SSAP selector
+ PSAP selector.

Data units:

Session Protocol Control Information (SPCI) is data passed between session entities to coordinate their joint operation. *Session User Data* (SUD) is passed between session entities on behalf of presentation entities. A *Session Protocol Data Unit* (SPDU) is a unit of data specified in the session protocol and consists of SPCI and possible SUD.

Session Interface Control Information (SICI) is passed between presentation and session entities to coordinate their joint operation. *Session Interface Data* (SID) is data passed from a presentation entity to a session entity for transmission to a remote presentation entity, or data passed from a session entity to a presentation entity after being received from a remote presentation entity. A *Session Service Data Unit* (SSDU) is a unit of data passed between two presentation entities whose integrity is to be maintained end-to-end. A *Session Interface Data Unit* (SIDU) is the unit of data passed across an SSAP in a single interaction and consists of SICI and possibly SID, which is the whole or part of an SSDU. Similar data units are defined for interactions between session and transport entities across a TSAP. They are:

Transport Interface Control Information (TICI),
Transport Interface Data (TID),
Transport Service Data Unit (TSDU),
Transport Interface Data Unit (TIDU).

Layer services:

A *service user* represents all those entities in an open system that make use of a service through a SAP. A presentation entity is therefore a session service user (SS-user) which uses session services through an SSAP. Similarly, a session entity is a transport service user (TS-user) which uses transport services through a TSAP.

A *service provider* is an abstract machine which models the behaviour of all those entities providing the service, as viewed by the user. SS-users therefore communicate by means of the session service provider (SS-provider) and TS-users communicate by means of the transport service provider (TS-provider).

Each service user interacts with the service provider by issuing or receiving *service primitives* at a SAP. The four types of service primitive are:

request (from user to provider),
indication (from provider to user),
response (from user to provider),
confirm (from provider to user).

2.2 The purpose of the session layer

The session layer is the fifth layer of the seven-layer OSI Reference Model. Broadly, its purpose is to add application-orientated services to the end-to-end communications channels provided by the transport layer. More specifically, it provides the means necessary for cooperating SS-users to organize and synchronize their dialogue and to manage their data exchange. To do this, the session layer provides services to:

a) establish a session connection between two SS-users;

b) support orderly data exchange interactions during the lifetime of the session connection; and

c) release the session connection.

The session service is provided by the session protocol using services available from the transport layer.

2.3 Services available from the transport layer

The transport layer provides TS-users in end open systems with a network-independent, transparent, data transfer service. It shields the higher layers from the technical details of how the communication is achieved. It optimizes the use of available network resources while maintaining, at minimum cost, a guaranteed quality of service required by the session entities.

2.3.1 Transport connection establishment

This service enables two TS-users to establish a transport connection between themselves. A transport connection is simply a full-duplex data path. The two session entities are identified by their unique TSAP addresses.

A TS-user may be associated with several transport connections simultaneously, to either the same or different TS-users. Both concurrent and consecutive transport connections are possible between two TS-users. Multi-endpoint transport connections are not allowed.

Each TS-user is provided with a TCEP identifier, enabling it to distinguish the new transport connection from all others accessible at its TSAP.

The quality of service required by the TS-users on the transport connection is negotiated between the TS-users and the TS-provider.

2.3.2 Normal data transfer

This service provides a reliable, full-duplex, transparent transfer of normal TSDUs over a transport connection, preserving TSDU boundaries, contents and sequence.

The agreed quality of service must be maintained by the TS-provider while the transport connection exists. If the TS-provider can no longer maintain that quality, it terminates the transport connection and notifies the TS-users of this fact.

This service is also subject to flow control, allowing receiving TS-users to control the rate at which sending TS-users may send data. The decision on when or how this flow control is applied is a local matter and is therefore not subject to any OSI specification.

2.3.3 Expedited data transfer

This service allows the transfer of small, expedited TSDUs over a transport connection. These TSDUs are transferred and/or processed with priority over normal TSDUs. This service is intended for signalling and interrupt purposes and may be used by either TS-user at any time that a transport connection exists.

2.3.4 Transport connection release

This service allows either TS-user to unconditionally release a transport connection and have the correspondent TS-user informed of the release.

2.4 Services provided by the session layer

2.4.1 Session connection establishment

This service enables two SS-users to establish a session connection, or *session*, between themselves, a process often called *binding*. The SS-user entities are identified by their unique SSAP addresses. The SS-users may negotiate and agree on a variety of options and parameters that may or may not be in effect for the session connection.

An SS-user may be associated with several session connections simultaneously, to either the same or different SS-users. Both concurrent and consecutive session connections are possible between two SS-users. Simultaneous session connection establishment requests may result in a corresponding number of session connections, but a session entity can always reject an incoming request. Multi-endpoint session connections are not allowed.

Each SS-user is provided with a SCEP identifier, enabling it to distinguish the new session connection from all others accessible at its SSAP.

2.4.2 Normal data transfer

This service allows a sending SS-user to transfer a normal SSDU to a receiving SS-user. This service also allows the receiving SS-user to ensure that it is not overloaded with data.

2.4.3 Expedited data transfer

This service allows the transfer of small, expedited SSDUs between SS-users. These SSDUs are transferred and/or processed with priority over normal SSDUs. This service is intended for signalling and interrupt purposes, and may be used by either SS-user at any time that a session connection exists.

2.4.4 Interaction management

This service allows the SS-users to control explicitly whose turn it is to exercise certain control functions. This service provides for:

a) *voluntary* exchange of the turn, where the SS-user which has the turn relinquishes it voluntarily, and

b) *forced* exchange of the turn, where, upon request from the SS-user which does not have the turn, the session service may force the SS-user with the turn to relinquish it. In this case, data may be lost.

This service enables SSDU exchange interactions to be either full-duplex (two-way simultaneous), half-duplex (two-way alternate) or simplex (one-way).

2.4.5 Session connection synchronization

This service allows SS-users to establish synchronization points in their dialogue. Once a synchronization point has been established and confirmed by the SS-users, both view the data transferred prior to the synchronization point as secured.

In the event of errors, this service then allows the SS-users to reset the session connection to a defined state and resume the dialogue from an agreed resynchronization point. The session layer is not responsible for any associated checkpointing or commitment action associated with resynchronization.

This service aids SS-users in recovering from communication failure without losing all the data already transferred - only unconfirmed data need be retransmitted.

2.4.6 Exception reporting

This service allows the SS-users to be notified of exceptional circumstances not covered by other services, such as unrecoverable SS-provider malfunctions or SS-user errors. It should be noted that the type of errors encountered in the session layer are procedural errors or failures of the underlying transport connection. Actual transmission errors are dealt with in the lower layers of the OSI Reference Model.

2.4.7 Session connection release

The session connection exists until released by either the SS-users or the session entities. This service allows SS-users to release a session connection in an *orderly* way without loss of data. It also allows either SS-user to request at any time that a session connection be *aborted*. In this case, data may be lost. The release of a session connection may also be initiated by one of the session entities supporting it.

2.5 Functions within the session layer

The functions within the session layer are those which must be performed by session entities in order to provide the session services. These functions are visible only within the session protocol and are therefore transparent to the presentation and transport layers. The major functions are described below:

2.5.1 Session address mapping

Generally, there is a many-to-one, hierarchical mapping between session and transport addresses. This does not imply multiplexing of session connections onto transport connections, but does imply that at session connection establishment time, more than one SS-user is a potential target of a session connection establishment request arriving on a given transport connection. The target SS-user is identified by the SSAP selector carried by the session protocol.

In many systems, a transport address may be used as the session address, i.e., there is a one-to-one mapping between the session and transport addresses.

2.5.2 Session connection mapping

To implement the transfer of data between the SS-users, the session connection is mapped onto, and uses, a transport connection. If a suitable transport connection is not available at session connection establishment time, one must be established. There is always a one-to-one mapping between session and transport connections. Conversely, there is no multiplexing or splitting between session and transport connections. A transport connection may, however, support several consecutive session connections.

To implement the mapping of a session connection onto a transport connection, the session layer must map SSDUs into SPDUs (possibly using the complementary operations of segmenting and reassembly), and SPDUs into TSDUs (possibly using concatenation and separation). SSDUs may not be mapped into SPDUs using blocking and deblocking.

2.5.3 Flow control

There is no peer flow control in the session layer. To prevent the receiving SS-user from being overloaded with data, it applies back pressure across the transport connection by using the transport flow control. However, this is a local matter and is therefore not subject to any OSI specification.

2.5.4 Expedited data transfer

The transfer of expedited SSDUs is generally accomplished by use of the transport expedited data service.

3. OVERVIEW OF THE X.400 MESSAGE HANDLING SYSTEM

The CCITT X.400 Message Handling System (MHS) is a generic, global, medium-independent, store-and-forward, electronic mail service. It standardizes electronic mail systems by integrating computer-based message systems with established, heterogeneous telematic services such as telex, teletex, facsimile, videotex, voice, etc. It allows its users to communicate by exchanging *messages*. Messages are comprised of addressing information and user content, which may include any combination of text, facsimile, graphics or other data structures.

The MHS is defined in CCITT Recommendations X.400 to X.430 [1] as a set of standard protocols, service definitions and user interfaces. These services and protocols reside in the application layer of the OSI Reference Model [2] and make extensive use of the Model's lower layers.

This section presents a general overview of the MHS, briefly describing its architecture and identifying its requirements of the lower layers of the OSI Reference Model. In particular, it shows why the session layer is of special interest to the MHS.

3.1 A functional model of the MHS

Figure 3.1 depicts a functional model of the MHS. This is followed by a brief description of the model's components and their functions.

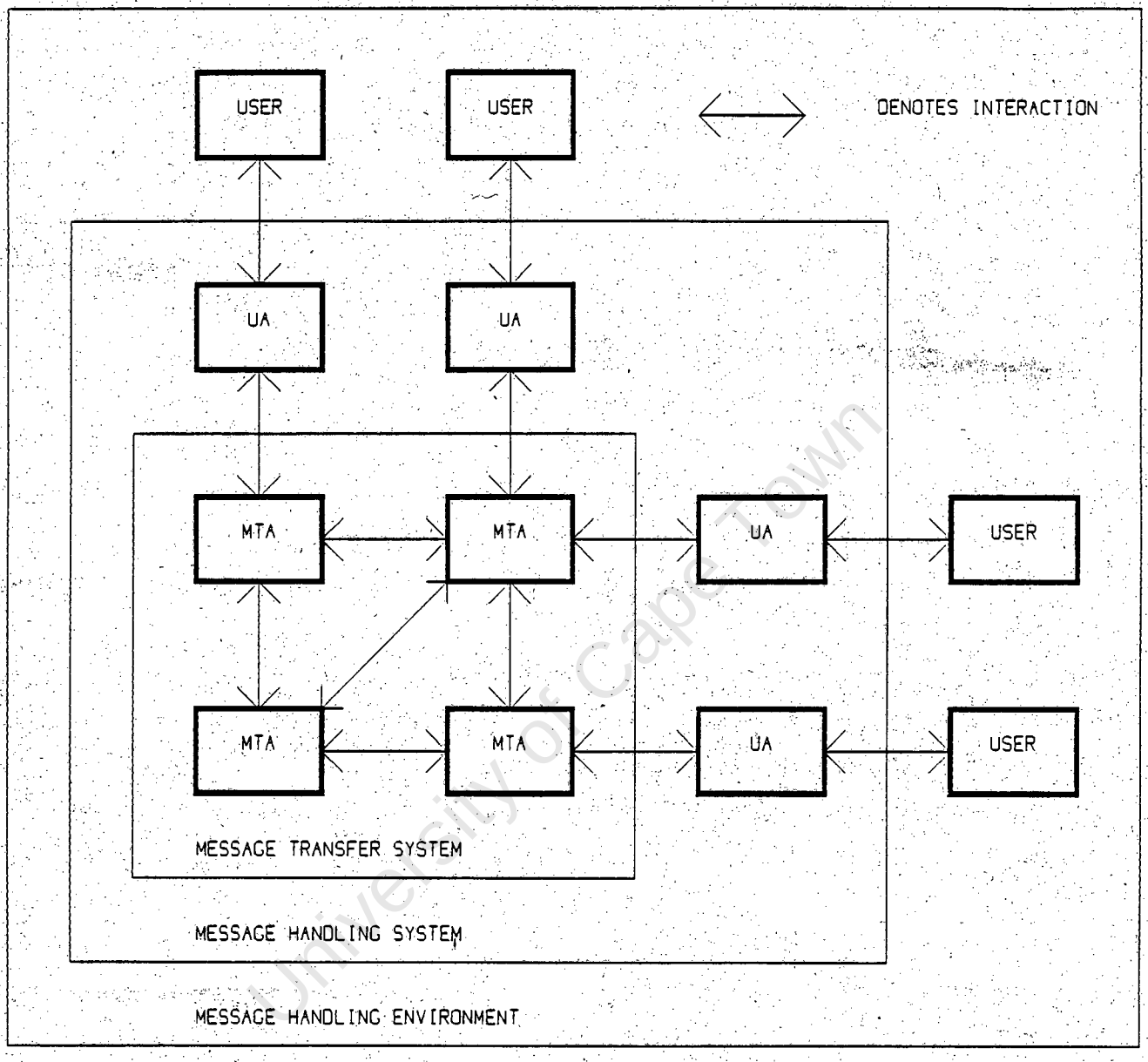

Figure 3.1 A functional model of the MHS

A *user* is either a person or a computer application. It is referred to as an *originator* (when sending a message) or a *recipient* (when receiving a message). An originator prepares a message with the assistance of its *User Agent* (UA). A UA is an application process that interacts with the *Message Transfer System* (MTS) to submit messages. The MTS delivers to one or more recipient UAs the messages submitted to it.

The MTS comprises a number of *Message Transfer Agents* (MTAs). Operating together, the MTAs relay messages and deliver them to the intended recipient UAs, which make the messages available to the intended recipients.

The collection of UAs and MTAs is called the *Message Handling System* (MHS). The MHS and all of its users are collectively referred to as the *Message Handling Environment*.

Messages consist of an *envelope* and *content*. The envelope carries addressing information used when transferring the message, while the content is the piece of information that the UA wishes delivered to one or more recipient UAs.

3.2 A layered model of the MHS

Figure 3.2 depicts a layered model of the MHS, showing how it fits into the OSI environment. This is followed by a brief description of the model's features.

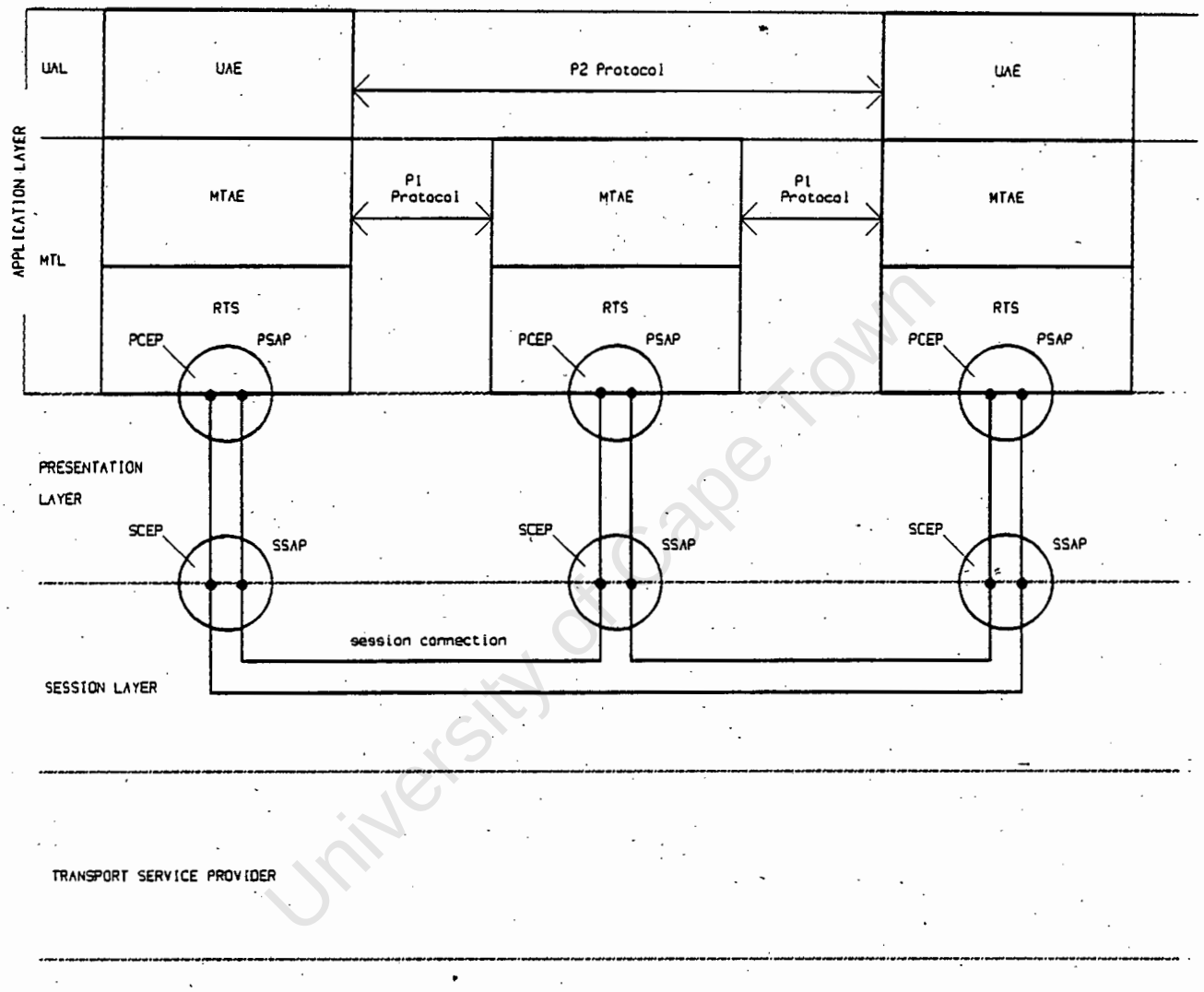

Figure 3.2 A layered model of the MHS

The MHS entities and protocols reside in the application layer of the OSI Reference Model. This allows the MHS application to use the underlying layers to establish network-independent connections between individual systems, and to establish session connections, permitting the MHS applications to reliably transfer messages between open systems.

The CCITT has split the application layer into two sublayers for the MHS application:

a) the *User Agent Layer* (UAL) contains the functionality associated with the contents of messages and is driven by the users;

b) the *Message Transfer Layer* (MTL) contains the MTA functionality and provides the MTS to the UAL.

The UA entity (UAE) embodies only those aspects of UA functionality associated with the operation of the UA to UA protocol (P2), not the local UA functionality. The MTA entity (MTAE) provides the functionality required to support the layer services of the MTL in cooperation with other MTAEs. The message transfer protocol (P1) is responsible for relaying messages between MTAEs and other interactions necessary to provide the MTL services.

A MTAE is divided into two elements: the *Message Dispatcher* and the *Reliable Transfer Server* (RTS). The Message Dispatcher performs the P1 protocol, which relies on the lower-level OSI protocols through the RTS. The RTS is therefore responsible for creating and maintaining connections between the MTAE and its peers, and for reliably transferring P1 Application Protocol Data Units (APDUs) by means of them. The RTS is therefore the subsystem in the MHS application layer that interfaces with the lower layers of the OSI Model. The detailed operation of the RTS and its use of the lower layers is described in CCITT Recommendation X.410 [9].

As will be shown in section 3.3.1, the RTS makes direct use of the session layer services. Since the OSI Reference Model permits direct interactions only between adjacent layers, the RTS cannot (strictly speaking) interact directly with session entities. This interaction is thus described as "through" the presentation layer which intervenes "transparently" to convey the interactions between the RTS and the session layer. This scheme is illustrated in Figure 3.2 as one-to-one mappings between PSAPs and SSAPs, and presentation "connections" and session connections. These mappings are consistent with the description of the presentation layer in the OSI Reference Model.

3.3 Use of layers below the application layer

The MHS APDUs are communicated between end systems by the operation of protocols in the lower six layers of the OSI Reference Model. Of particular interest to the MHS are the presentation and session layers.

3.3.1 The presentation layer

In general, the presentation layer serves to determine a common transfer syntax for the exchange of APDUs between application entities. Where the syntax preferred or used by each application entity differs, the presentation layer would provide a translation or mapping of one syntax to another, or each into a common transfer syntax.

This function of syntax conversion is particularly important in MH, as one of the features of MH is its automatic conversion of user data of one kind (e.g. text), to data of another kind (e.g. facsimile). A presentation protocol could be used to determine conversion requirements, and relieve the application protocol of these considerations.

However, the development of suitable protocols and conversion algorithms for a general presentation layer have lagged the work - and requirements - of MH by some years. In order to achieve functioning MH protocols in time for the 1984 CCITT Plenary, MH was designed to bypass entirely the presentation layer. The presentation layer is being defined to permit this type of approach, in which application entities can access the session services directly, so MH remains consistent with the OSI Reference Model.

3.3.2 The session layer

In the absence of a presentation layer protocol in MH systems, the session layer is directly responsible for the secure delivery of MH APDUs. This means that the Reliable Transfer Server (RTS) actually interacts directly with the session layer.

MH uses the connection-orientated session service defined in CCITT Recommendation X.215 [3], which employs the session protocol specified in CCITT Recommendation X.225 [4]. The detailed use of the session service by MH is the subject of the next section, section 4.

3.3.3 The transport layer and below

MH uses the connection-orientated transport service defined in CCITT Recommendation X.214 [10], which employs the transport protocol specified in CCITT Recommendation X.224 [11].

Protocols below the transport layer are network-dependent and are therefore not specified for MH. MH systems can thus be implemented over any telecommunications network providing adequate quality of service. In practice, MH

would often be implemented on a packet-switched network with CCITT X.25 the appropriate protocol.

4. THE SESSION SERVICE FOR X.400

The session service defines in an abstract, conceptual, implementation-independent way the essential properties and features of the service provided by the session layer to its users. It is defined in terms of:

a) the primitive events and actions of the service;
b) the parameter data associated with each primitive action and event;
c) the relationships between, and the valid sequences of the actions and events.

The general session layer provides a great many optional services to SS-users. This enables it to support all possible application types. However, the CCITT X.400 application represents only a subset of all possible application types, and therefore requires only a subset of the general session services.

Identifying this subset of session services required by X.400 is the subject of this section. It specifies precisely which of the general session services are used by X.400 and how X.400 uses them. This information is required by section 5, which will show how the general session protocol may be tailored to provide only these services required by X.400.

Section 3 identified the RTS as that element of an X.400 application entity which interfaces directly with the session layer. The RTS is therefore an X.400 SS-user. This section derives the session service for the RTS by applying the RTS's session service requirements, as specified in CCITT Recommendation X.410 section 4 [9], to the general Session Service Definition of CCITT Recommendation X.215 [3].

4.1 Definition of terms

For the purpose of this section and the rest of this thesis, the following definitions apply:

calling SS-user
An SS-user that initiates a session connection establishment request.

called SS-user
An SS-user with whom a calling SS-user wishes to establish a session connection.

Note: Calling SS-users and called SS-users are defined with respect to a single connection. An SS-user can be both a calling and a called SS-user simultaneously.

sending SS-user
An SS-user that sends data during the data transfer phase of a session connection.

receiving SS-user
An SS-user that receives data during the data transfer phase of a session connection.

requestor; requesting SS-user
An SS-user that initiates a particular action.

acceptor; accepting SS-user
An SS-user that accepts a particular action.

conditional parameter
A parameter whose presence in a request or response primitive depends on some condition; and whose presence in an indication or confirm primitive is mandatory if it was present in the preceding primitive, or absent if it was absent in the preceding primitive.

proposed parameter
The value for a parameter proposed by an SS-user, in a session connection request or response primitive, that it wishes to use on the session connection.

selected parameter
The value for a parameter that has been chosen for use on the session connection.

4.2 Model of the session service

Figure 4.1 depicts a model of the session service. It shows the SS-provider providing a session connection between two correspondent SS-users. Each SS-user accesses session services at its SSAP, in which the session connection is terminated by a SCEP.

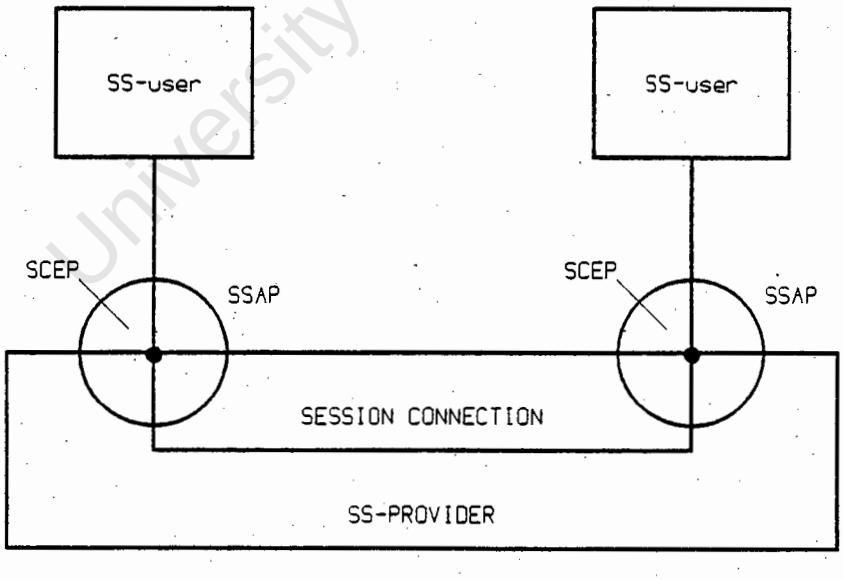

Figure 4.1 Model of the session service

4.3 The token concept

A token is an attribute of a session connection which is dynamically assigned to one SS-user at a time to permit certain services to be invoked. It controls the right to exclusive use of the service.

Four tokens are defined, each controlling the use of one or more services. Table 4.1 lists the tokens, their standard abbreviated names and the services controlled by each.

Table 4.1 Tokens

token	abbrv	services controlled
data	dk	normal data transfer
release	tr	orderly connection release
synchronize-minor	mi	minor synchronization
major/activity	ma	major synchronization activity management

A token is always in one of the following states:

a) *available*, in which case it is always:

   1) *assigned* to one SS-user (the *owner* of the token), who then has the exclusive right to use the associated service (provided that no other restrictions apply);

   2) *not assigned* to the other SS-user, who does not have the right to use the service but may acquire it later;

b) *not available* to either SS-user, in which case neither SS-user has the exclusive right to use the associated service. The service then becomes inherently available to both SS-users (data transfer and orderly release), or unavailable to both SS-users (synchronization and activity management).

The RTS restricts its use of the tokens as follows:

a) The data, synchronize-minor and major/activity tokens are always available. The release token is never available.

b) The tokens are never separated, i.e., all the available tokens are always assigned to one of the correspondent RTSs. The owner of the tokens is referred to as the *sending* RTS, the other as the *receiving* RTS.

Subsection 4.8.1 shows how specific tokens are made available to a session connection, while 4.10.2 defines the token restrictions placed on session services.

4.4 The major synchronization and activity concepts

Certain session services allow the SS-users to partition parts of their dialogue. The SS-users may 'mark' points in their dialogue, upon which the session layer ensures complete separation of the dialogue before and after the mark. This can be done by using either "Major Synchronization Points" or "Activities".

These two styles for SS-user dialogue separation, and corresponding resynchronization procedures, stem from the fact that a major concern during the development of the session layer standards has been the issue of compatibility with existing CCITT standards. As a result, "Activities" are intended for use by CCITT applications, while "Major

Synchronization Points" are intended for ECMA applications. Since X.400 is a CCITT application, the RTS uses the activity concept for separating its dialogues.

A brief description of both these concepts is presented below, showing why the activity concept is better suited to RTS requirements than the major synchronization concept.

4.4.1 Major synchronization

Major synchronization points are intended for separating general session dialogues which use full-duplex normal and expedited data exchange.

SS-users may insert major synchronization points into the data they are transmitting, each identified by a serial number maintained by the SS-provider. Any semantics which SS-users may give to their major synchronization points are transparent to the SS-provider.

Major synchronization points structure the data exchange into a series of *dialogue units*. The characteristic of a dialogue unit is that all communication within it is completely separated from all communication before and after it. A major synchronization point indicates the end of one dialogue unit and the start of the next. Each major synchronization point must be confirmed explicitly.

An example of using major synchronization points would be to indicate a change in application dialogue context, e.g. from file transfer to message-passing.

Major synchronization is not suited to RTS requirements because, as will be shown, the RTS never uses full-duplex or expedited data exchanges. In addition, RTS dialogue context remains constant - that of Message Handling.

4.4.2 Activities

Activities are intended for separating half-duplex data exchanges.

The activity concept allows SS-users to distinguish between different logical pieces of work called *activities*. Each activity may be structured into one or more dialogue units by use of major synchronization points. Only one activity is allowed on a session connection at a time, but there may be several consecutive activities during a session connection. An activity can be interrupted and then resumed on the same or on a subsequent session connection. This can be considered as a form resynchronization. The SS-users may transfer only *capability* data outside an activity.

An example of using activities would be the separation of documents on a document transfer connection.

Activities are suited to RTS requirements because RTS dialogue is always half-duplex.

4.5 The minor synchronization point concept

Minor synchronization points are intended for partitioning simplex data flow with no use of session expedited data.

SS-users may insert minor synchronization points into the data they are transmitting. Each minor synchronization point is identified by a serial number maintained by the SS-provider. Any semantics which SS-users may give to their minor synchronization points are transparent to the SS-provider. Each minor synchronization point may or may not be explicitly confirmed.

Minor synchronization points are used to structure data within either dialogue units or activities. Figure 4.2 shows how an RTS activity may be structured through the use of minor synchronization points.

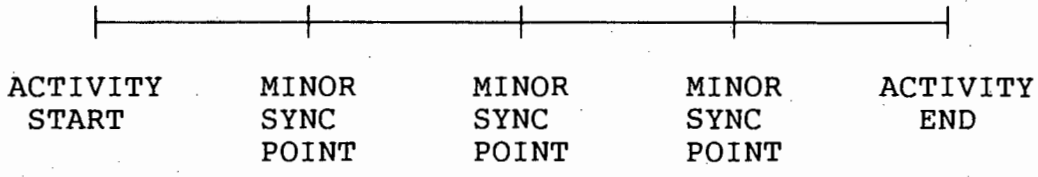

Figure 4.2 Example of a structured activity

4.6 The resynchronization concept

Resynchronization may be initiated by either SS-user. It sets the session connection to a defined state, reassigns the tokens, sets the synchronization point serial number to a new value and purges all undelivered data. Resynchronization is never used by the RTS. Instead, the RTS uses a form of resynchronization associated with activities.

4.7 Phases and services of the general session service

The general session service comprises three phases, each of which provides certain services. The purpose of each phase and a brief description of the associated services is given here. Of these services, those used by the RTS are identified.

4.7.1 The session connection establishment phase

This phase is concerned with establishing a session connection between two SS-users. It has one service associated with it:

a) The **Session Connection** service.
This service is always provided to all SS-users. It enables two SS-users to establish a session connection between themselves. Simultaneous attempts by both SS-users to establish a connection may result in two session connections. An SS-user may always reject an unwanted connection. No architectural restriction is placed on the number of concurrent session connections associated with an SS-user, or between two SS-users.

The SS-users may negotiate the values of various session connection parameters. By the end of the session connection establishment phase, the SS-users have agreed on a set of parameter values concerning the session connection.

4.7.2 The data transfer phase

This phase is concerned with the exchange of data between the two SS-users connected in the session connection establishment phase.

There are four services concerned with data transfer:

a) The **Normal Data Transfer** service.
This service is always provided on every session connection. It allows SS-users to exchange unconfirmed, unlimited-length, normal SSDUs over a session connection. The SS-provider delivers each normal SSDU to the SS-user as soon as possible. Use of this service is controlled by the data token if it is available.

b) The **Expedited Data Transfer** service.
This optional service allows SS-users to exchange unconfirmed, limited-length, expedited SSDUs over a session connection, free from the token and flow control constraints of the other three data transfer services. This service is not used by the RTS.

c) The **Typed Data Transfer** service.
Generally, SS-user data consists of two distinct types: application user data and PDUs from Layers 6 and 7. For some applications, the latter should not be restricted to the token control which is exerted over the former. Token control exists to manage the dialogue between two applications, but correct functioning of Layers 6 and 7, especially during error or recovery situations, may well require unrestricted protocol exchanges.

Thus, the optional Typed Data Transfer service is provided. It allows unconfirmed, unlimited-length, Typed SSDUs to be exchanged regardless of the availability and assignment of the data token. Typed data is subject to the same flow control as normal data, so if one is blocked, the other is also blocked.

When both typed and normal data are blocked, only expedited data can be passed.

This service is unnecessary for the RTS because RTS SS-user data *is* always application user data. Also, no unrestricted protocol exchanges are ever required between two RTSs.

d) The **Capability Data Exchange** service.
This optional service allows SS-users to exchange a limited amount of confirmed data while not within an activity. It is provided solely as a means for SS-users to exchange information about their "capability" to participate in a new activity. This service is not used by the RTS because all RTS data exchanges occur within activities.

There are three services concerned with token management:

e) The **Give Tokens** service.
This optional service allows an SS-user to surrender one or more specific tokens to the other SS-user. This service is unnecessary for the RTS because the Give Control service satisfies all RTS token transfer needs.

f) The **Please Tokens** service.
This optional service allows an SS-user to request the other SS-user to transfer one or more specific tokens to it. Since the RTS does not separate the tokens, it uses this service to request all the available tokens.

g) The **Give Control** service.
This optional service allows an SS-user to surrender all available tokens to the other SS-user. Since the RTS does not separate the tokens, this service satisfies all its token transfer needs.

There are three services concerned with synchronization and resynchronization:

h) The **Minor Synchronization Point** service.
This optional service allows the SS-users to separate the flow of one-way normal and typed SSDUs transmitted before the service was invoked from the subsequent flow of normal and typed SSDUs. To do this, it enables SS-users to define minor synchronization points in the flow of SSDUs. These minor synchronization points may optionally be confirmed, but have no implications on the data flow.

Minor synchronization points are identified by synchronization point serial numbers. The serial number is incremented by one each time a minor synchronization point is placed in the data flow, and each time a minor synchronization point is received, so that both SS-users have the same serial numbers for the same synchronization point. Use of this service is controlled by the synchronize-minor token.

This service is very important to the RTS. RTS APDUs can be very long, because an entire (even multi-page) user message is carried in a single APDU. The use of synchronization points within these APDUs minimizes the retransmission required after errors and is therefore important in keeping down transmission overheads.

i) The **Major Synchronization Point** service.
This optional service allows the SS-users to confine the flow of full-duplex, sequentially transmitted normal, typed and expedited SSDUs in each direction within a dialogue unit. To do this, it enables SS-users to define major synchronization points in the flow of SSDUs. A major synchronization point must be confirmed before the requesting SS-user is permitted to send any subsequent data, thereby clearly separating the data flow before and after the major synchronization point into dialogue units. Use of this service is controlled by the major/activity token.

This service is not required by the RTS because the RTS does not use full-duplex data transmission, typed or expedited data. Also, the RTS does not need to structure its dialogue into dialogue units because RTS dialogue context remains constant, that of Message Handling.

j) The **Resynchronize** service.
This optional service allows the SS-users to re-establish communication, in an orderly manner, within the current session connection. This typically occurs following an error, lack of response by either SS-user or the SS-provider, or disagreements between SS-users. This service sets the session connection to either a previous or a new synchronization point and reassigns the available tokens. This service may cause loss of normal, typed and expedited SSDUs.

This service is not used by the RTS. Instead, it uses the form of resynchronization provided by the activity management services.

There are two services concerned with reporting errors or unanticipated situations:

k) The **Provider-Initiated Exception Reporting** service. This optional service allows the SS-provider to notify both SS-users that a service cannot be completed due to SS-provider protocol errors or exception conditions which are not covered by other services. These exception conditions are less severe than those requiring abort of the session connection and the SS-provider anticipates that the SS-users will be able to overcome the problem. This service may cause loss of normal, typed and expedited SSDUs.

   This service is not used by the RTS. Instead, the RTS assumes that the SS-provider will abort a session connection upon detecting any unrecoverable error.

l) The **User-Initiated Exception Reporting** service. This optional service allows an SS-user to report an exception condition when the data token is available but not assigned to the SS-user. By initiating this service, the sending SS-user indicates a problem less severe than one requiring abort of the session connection. The sending SS-user anticipates that the receiving SS-user can overcome this problem and allows it to take the most appropriate course of action. This service may cause loss of normal, typed and expedited SSDUs. This service is used by the RTS.

There are five optional services concerned with activities. All these services are controlled by the major/activity token:

m) The **Activity Start** service.
This service allows an SS-user to indicate the start of a new activity.

n) The **Activity Resume** service.
This service allows an SS-user to indicate that a previously interrupted activity is re-entered.

o) The **Activity Interrupt** service.
This service allows an SS-user to abnormally terminate an activity with the implication that the work so far achieved is not to be discarded and may be resumed later. This service may cause loss of undelivered normal, typed and expedited SSDUs.

p) The **Activity Discard** service.
This service allows an SS-user to abnormally terminate an activity with the implication that the work so far achieved is to be discarded (not controlled by the SS-provider), and not resumed. This service may cause loss of undelivered normal, typed and expedited SSDUs.

q) The **Activity End** service.
This service allows an SS-user to indicate the normal end of an activity.

These activity management services are very important to X.400 because they provide reliability of data transfer.

X.400 APDUs are transferred within activities. Since the local session entity confirms delivery of activities, the delivery of X.400 APDUs are implicitly confirmed. This allows many X.400 application protocol elements to be unconfirmed, leading to simple, efficient X.400 application protocols.

Using the activity services may lead to a state where no activity is in progress on the session connection. When the RTS enters this state, it may invoke only the following services:

Activity Start,
Activity Resume,
Please Tokens,
Give Control,
Normal Data Transfer,
User Abort, and
Orderly Release.

4.7.3 The session connection release phase

This phase is concerned with releasing a previously established session connection. It has three services associated with it:

a) The **Orderly Release** service.
This service is always provided on all session connections. It allows either SS-user to release a session connection in an orderly manner. This is done cooperatively between the two SS-users without loss of data, after all in-transit data has been delivered and accepted by both SS-users.

b) The **User-Initiated Abort** service.
This service is always provided on all session connections. It allows either SS-user to initiate immediate release of a session connection and have the other SS-user informed of the release. This service terminates any outstanding service request and causes loss of all undelivered data.

c) The **Provider-Initiated Abort** service.
This service is always provided on all session connections. It allows the SS-provider to indicate to both SS-users the immediate release of a session connection for internal reasons. This service terminates any outstanding service request and causes loss of all undelivered data.

4.8 Functional units and subsets

4.8.1 Functional units

Since there are a great many optional services provided by the session layer, and as the set of services needed will vary from application to application, so many session layer implementations will only implement the subset of services needed to support the applications that have been implemented. By negotiating the set of services that are needed for the connection at establishment time, the situation is avoided whereby a connection is established and only later is it discovered that it cannot be used.

SS-user service requirements are negotiated during the session connection establishment phase in terms of functional units. Functional units are logical groupings of related services.

Certain functional units require the availability of a particular token. This implies that the functional units selected for use on a session connection determine which tokens are available on that connection. Functional units requiring a token also include those services necessary to request and transfer that token.

Table 4.2 lists all the functional units. For each it specifies its standard abbreviated name, the services associated with it, the tokens associated with it and pre-requisite functional units that must be selected with it. The last column indicates which functional units are mutually-exclusive, i.e., which functional units may not be selected together on the same session connection. Those functional units, services and tokens used by the RTS are indicated in **bold** font.

```
{
 Build Initial serial number
}

PURE PROCEDURE Build23PIU(VAR tsdu     : TSDUTYPE;
                          InitialSpsn : INTEGER);

VAR PI                          : ByteTYPE;
    LI,factor,index,digpos,dig : INTEGER;
    zero                        : BOOLEAN;

BEGIN
  PI := 23;

  IF InitialSpsn > 0
  THEN
    LI := TRUNC(LN(InitialSpsn)/LN(10)) + 1;
  ELSE
    LI := 1;

  IF NOT FU(ACT) AND (FU(SY) OR FU(MA) OR FU(RESYN))
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);
      factor := 100000;
      index  := 1;
      zero   := TRUE;

      FOR digpos := 5 DOWNTO 0 DO
        BEGIN
          dig := InitialSpsn DIV factor;
          IF (dig <> 0) OR NOT zero
          THEN
            BEGIN
              zero := FALSE;
              tsdu.d[tsdu.l+index] := dig + 48;
              index := index + 1;
            END;
          InitialSpsn := InitialSpsn MOD factor;
          factor := factor DIV 10;
        END;

      IF zero
      THEN
        tsdu.d[tsdu.l+1] := 0 + 48;

      tsdu.l := tsdu.l + LI;
    END;
END;
```

```
{
 Build Enclosure item
}

PURE PROCEDURE Build25PIU(VAR tsdu      : TSDUTYPE;
                          EnclosureItem : EnclosureItemTYPE);

VAR PI : ByteTYPE;
    LI : INTEGER;

BEGIN
  PI := 25;
  LI := 1;

  IF EnclosureItem <> BEGIN_END {value 3}
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);
      tsdu.d[tsdu.l+1] := ORD(EnclosureItem); {values 0,1 or 2}
      tsdu.l := tsdu.l + LI;
    END;
END;
```

```
{
 Build Token setting item
}

PURE PROCEDURE Build26PIU(VAR tsdu          : TSDUTYPE;
                          TokenSettingItem : InitialTokensTYPE);

VAR PI              : ByteTYPE;
    LI,byte,factor : INTEGER;
    token           : TokenTYPE;
    SomeAvail       : BOOLEAN;

BEGIN
  PI := 26;
  LI := 1;

  FORONE token : TokenTYPE
    SUCHTHAT AV(token) DO
      SomeAvail := TRUE;
    OTHERWISE
      SomeAvail := FALSE;

  IF SomeAvail
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);
      factor := 1;
      byte   := 0;

      FOR token := DKT TO TRT DO
        BEGIN

          CASE TokenSettingItem[token] OF
            REQUESTOR_SIDE   : byte := byte + 0 * factor;
            ACCEPTOR_SIDE    : byte := byte + 1 * factor;
            ACCEPTOR_CHOOSES : byte := byte + 2 * factor;
          END;

          factor := factor * 4;
        END;

        tsdu.d[tsdu.l+1] := byte;
        tsdu.l := tsdu.l + LI;
    END;
END;
```

```
{
 Build Activity identifier
}

PURE PROCEDURE Build41PIU(VAR tsdu    : TSDUTYPE;
                          AcitvityId : Bytes6TYPE);

VAR PI    : ByteTYPE;
    LI,i : INTEGER;

BEGIN
  PI := 41;
  LI := ActivityId.l;

  IF LI > 0
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);

      FOR i := 1 TO LI DO
        tsdu.d[tsdu.l+i] := ActivityId.d[i];

      tsdu.l := tsdu.l + LI;
    END;
END;
```

```
{
 Build Serial number
}

PURE PROCEDURE Build42PIU(VAR tsdu : TSDUTYPE;
                          spsn     : INTEGER);

VAR PI                          : ByteTYPE;
    LI,factor,index,digpos,dig : INTEGER;
    zero                        : BOOLEAN;

BEGIN
  PI := 42;

  IF spsn > 0
  THEN
    LI := TRUNC(LN(spsn)/LN(10)) + 1;
  ELSE
    LI := 1;

  IF TRUE
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);
      factor := 100000;
      index  := 1;
      zero   := TRUE;

      FOR digpos := 5 DOWNTO 0 DO
        BEGIN
          dig := spsn DIV factor;
          IF (dig <> 0) OR NOT zero
          THEN
            BEGIN
              zero := FALSE;
              tsdu.d[tsdu.l+index] := dig + 48;
              index := index + 1;
            END;
          spsn   := spsn MOD factor;
          factor := factor DIV 10;
        END;

      IF zero
      THEN
        tsdu.d[tsdu.l+1] := 0 + 48;

      tsdu.l := tsdu.l + LI;
    END;
END;
```

```
{
 Build User data
}

PURE PROCEDURE Build46PIU(VAR tsdu    : TSDUTYPE;
                          SSuserData : Bytes512TYPE);

VAR PI    : ByteTYPE;
    LI,i : INTEGER;

BEGIN
  PI := 46;
  LI := SSuserData.l;

  IF LI > 0
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);

      FOR i := 1 TO LI DO
        tsdu.d[tsdu.l+i] := SSuserData.d[i];

      tsdu.l := tsdu.l + LI;
    END;
END;


{
 Build Reflect parameter values
}

PURE PROCEDURE Build49PIU(VAR tsdu           : TSDUTYPE;
                          ReflectParameters : Bytes9TYPE);

VAR PI    : ByteTYPE;
    LI,i : INTEGER;

BEGIN
  PI := 49;
  LI := ReflectParameters.l;

  IF LI > 0
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);

      FOR i := 1 TO LI DO
        tsdu.d[tsdu.l+i] := ReflectParameters.d[i];

      tsdu.l := tsdu.l + LI;
    END;
END;
```

```
{
 Build Reason code
}

PURE PROCEDURE Build50PIU(VAR tsdu   : TSDUTYPE;
                          ReasonCode : ReasonCodeTYPE);

VAR PI,byte : ByteTYPE;
    LI,i    : INTEGER;

BEGIN
  PI := 50;
  LI := ReasonCode.Data.l + 1;

  IF TRUE
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);

      CASE ReasonCode.Reason OF
        SSU_UNSPECIFIED       : byte := 0;
        SSU_CONGESTED         : byte := 1;
        SSU_SEE_DATA          : byte := 2;
        SEQUENCE_ERROR        : byte := 3;
        LOCAL_SSU_ERROR       : byte := 5;
        PROCEDURE_ERROR       : byte := 6;
        DEMAND_DK             : byte := 128;
        CALLED_SSAP_UNKNOWN   : byte := 129;
        CALLED_SSU_UNATTACHED : byte := 130;
        SSP_CONGESTED         : byte := 131;
        PROPOSED_PROTOCOL     : byte := 132;
      END;

      tsdu.d[tsdu.l+1] := byte;

      IF LI > 1
      THEN
        FOR i := 1 TO LI-1 DO
          tsdu.d[tsdu.l+i+1] := ReasonCode.Data.d[i];

      tsdu.l := tsdu.l + LI;
    END;
END;
```

```
{
 Build Calling SSAP identifier
}

PURE PROCEDURE Build51PIU(VAR tsdu       : TSDUTYPE;
                          CallingSSAPid : Bytes16TYPE);

VAR PI    : ByteTYPE;
    LI,i : INTEGER;

BEGIN
  PI := 51;
  LI := CallingSSAPid.l;

  IF LI > 0
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);

      FOR i := 1 TO LI DO
        tsdu.d[tsdu.l+i] := CallingSSAPid.d[i];

      tsdu.l := tsdu.l + LI;
    END;
END;


{
 Build Called SSAP identifier
}

PURE PROCEDURE Build52PIU(VAR tsdu      : TSDUTYPE;
                          CalledSSAPid : Bytes16TYPE);

VAR PI    : ByteTYPE;
    LI,i : INTEGER;

BEGIN
  PI := 52;
  LI := CalledSSAPid.l;

  IF LI > 0
  THEN
    BEGIN
      BuildHeader(tsdu,PI,LI);

      FOR i := 1 TO LI DO
        tsdu.d[tsdu.l+i] := CalledSSAPid.d[i];

      tsdu.l := tsdu.l + LI;
    END;
END;
```

```
{
 PROCEDURE: PGIU building procedures

            Given PGIU parameter values and a TSDU, each of these
            procedures appends a unique PGIU, as part of a SPDU,
            onto the TSDU.

            These procedures only build those PGIUs forming part
            of those SPDUs required by X.400.

 INPUTS:    tsdu       - the TSDU. Its .l field indicates its
                         current length.

            parameters - the PGIU parameter values.

 OUTPUTS:   tsdu.l - is updated to include the appended PGIU,
                     if any.

 CALLS:     BuildHeader, AppendTSDU, PIU building procedures.
}
```

```
{
 Build Connection identifier for CN SPDU
}

PURE PROCEDURE BuildlaPGIU(VAR tsdu          : TSDUTYPE;
                           CallingSSuserRef  : Bytes64TYPE;
                           CommonRef         : Bytes64TYPE;
                           AdditionalRef     : Bytes4TYPE);

VAR PGI   : ByteTYPE;
    LI    : INTEGER;
    pgiu  : TSDUTYPE;

BEGIN
  pgiu.l := 0;
  Buildl0PIU(pgiu,CallingSSuserRef);
  Buildl1PIU(pgiu,CommonRef);
  Buildl2PIU(pgiu,AdditionalRef);

  PGI := 1;
  LI  := pgiu.l;

  IF LI > 0
  THEN
    BEGIN
      BuildHeader(tsdu,PGI,LI);
      AppendTSDU(tsdu,pgiu);
   END;
END;
```

```
{
 Build Connection identifier for AC, RF SPDUs
}

PURE PROCEDURE Buildl6PGIU(VAR tsdu          : TSDUTYPE;
                           CalledSSuserRef : Bytes64TYPE;
                           CommonRef       : Bytes64TYPE;
                           AdditionalRef   : Bytes4TYPE);

VAR PGI  : ByteTYPE;
    LI   : INTEGER;
    pgiu : TSDUTYPE;

BEGIN
  pgiu.l := 0;
  Build9PIU(pgiu,CalledSSuserRef);
  Buildl1PIU(pgiu,CommonRef);
  Buildl2PIU(pgiu,AdditionalRef);

  PGI := 1;
  LI  := pgiu.l;

  IF LI > 0
  THEN
    BEGIN
      BuildHeader(tsdu,PGI,LI);
      AppendTSDU(tsdu,pgiu);
    END;
END;
```

```
{
 Build Connect/Accept item
}

PURE PROCEDURE Build5PGIU(VAR tsdu           : TSDUTYPE;
                          ProtocolOptions    : ByteTYPE;
                          maxTSDUlen0        : INTEGER;
                          maxTSDUlen1        : INTEGER;
                          VersionNumber      : ByteTYPE;
                          InitialSpsn        : INTEGER;
                          TokenSettingItem   : InitialTokensTYPE);

VAR PGI  : ByteTYPE;
    LI   : INTEGER;
    pgiu : TSDUTYPE;

BEGIN
  pgiu.l := 0;
  Build19PIU(pgiu,ProtocolOptions);
  Build21PIU(pgiu,maxTSDUlen0,maxTSDUlen1);
  Build22PIU(pgiu,VersionNumber);
  Build23PIU(pgiu,InitialSpsn);
  Build26PIU(pgiu,TokenSettingItem);

  PGI := 5;
  LI  := pgiu.l;

  IF LI > 0
  THEN
    BEGIN
      BuildHeader(tsdu,PGI,LI);
      AppendTSDU(tsdu,pgiu);
    END;
END;
```

```
{
 Build Linking information
}

PURE PROCEDURE Build33PGIU(VAR tsdu          : TSDUTYPE;
                           CalledSSuserRef   : Bytes64TYPE;
                           CallingSSuserRef  : Bytes64TYPE;
                           CommonRef         : Bytes64TYPE;
                           AdditionalRef     : Bytes4TYPE;
                           OldActivityId     : Bytes6TYPE;
                           spsn              : INTEGER);

VAR PGI  : ByteTYPE;
    LI   : INTEGER;
    pgiu : TSDUTYPE;

BEGIN
  pgiu.l := 0;
  Build9PIU(pgiu,CalledSSuserRef);
  Build10PIU(pgiu,CallingSSuserRef);
  Build11PIU(pgiu,CommonRef);
  Build12PIU(pgiu,AdditionalRef);
  Build41PIU(pgiu,OldActivityId);
  Build42PIU(pgiu,spsn);

  PGI := 33;
  LI  := pgiu.l;

  IF LI > 0
  THEN
    BEGIN
      BuildHeader(tsdu,PGI,LI);
      AppendTSDU(tsdu,pgiu);
    END;
END;
```

```
{
 Build User data
}

PURE PROCEDURE Build193PGIU(VAR tsdu    : TSDUTYPE;
                            SSuserData : Bytes512TYPE);

VAR PGI  : ByteTYPE;
    LI,i : INTEGER;

BEGIN
  PGI := 193;
  LI  := SSuserData.l;

  IF LI > 0
  THEN
    BEGIN
      BuildHeader(tsdu,PGI,LI);
      FOR i := 1 TO LI DO
        tsdu.d[tsdu.l+i] := SSuserData.d[i];
      tsdu.l := tsdu.l + LI;
    END;
END;



{
 Build User information field
}

PURE PROCEDURE BuildUserInfo(VAR tsdu : TSDUTYPE;
                             UserInfo : SSDUTYPE);

VAR LI,i : INTEGER;

BEGIN
  LI := UserInfo.l;

  IF LI > 0
  THEN
    BEGIN
      FOR i := 1 TO LI DO
        tsdu.d[tsdu.l+i] := UserInfo.d[i];
      tsdu.l := tsdu.l + LI;
    END;
END;
```

```
{
 PROCEDURE: SPDU building procedures

            Given SPDU parameter values and a TSDU, each of these
            procedures builds a unique SPDU onto the TSDU.

            These procedures only build those SPDUs required
            by X.400.

            Category 0 & 1 SPDUs are built from the 1st TSDU byte.
            Category 2 SPDUs are preceded by either a GT or PT
            SPDU according to the rules of Basic Concatenation.
            Since the X.400 SPM makes no real use of this feature,
            such GT and PT SPDUs have no parameter fields and are
            therefore reduced to their headers only.

 INPUTS:    tsdu       - the TSDU.
            parameters - the SPDU parameter values.

 OUTPUTS:   tsdu.1 - is set to include the built SPDU(s).

 CALLS:     BuildHeader, AppendTSDU,
            PIU and PGIU building procedures.
}
```

```
{
 Build DATA TRANSFER SPDU - category 2
}

PURE PROCEDURE BuildDT(VAR tsdu       : TSDUTYPE;
                       EnclosureItem : EnclosureItemTYPE;
                       UserInfo      : SSDUTYPE);

VAR SI   : ByteTYPE;
    LI   : INTEGER;
    spdu : TSDUTYPE;

BEGIN
  tsdu.l := 0;
  spdu.l := 0;

  Build25PIU(spdu,EnclosureItem);

  SI := 1;
  LI := spdu.l;

  BuildUserInfo(spdu,UserInfo);

  BuildHeader(tsdu,1,0);   {GT SPDU}
  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;


{
 Build PLEASE TOKENS SPDU - category 0
}

PURE PROCEDURE BuildPT(VAR tsdu    : TSDUTYPE;
                       TokenItem  : TokenSetTYPE;
                       SSuserData : Bytes512TYPE);

VAR SI   : ByteTYPE;
    LI   : INTEGER;
    spdu : TSDUTYPE;

BEGIN
  tsdu.l := 0;
  spdu.l := 0;

  Build16PIU(spdu,TokenItem);

  Build193PGIU(spdu,SSuserData);

  SI := 2;
  LI := spdu.l;

  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;
```

```
{
 Build FINISH SPDU - category 1
}

PURE PROCEDURE BuildFN(VAR tsdu    : TSDUTYPE;
                       TCdis       : TCdisTYPE;
                       SSuserData : Bytes512TYPE);

VAR SI   : ByteTYPE;
    LI   : INTEGER;
    spdu : TSDUTYPE;

BEGIN
  tsdu.l := 0;
  spdu.l := 0;

  Build17PIU(spdu,TCdis);

  Build193PGIU(spdu,SSuserData);

  SI := 9;
  LI := spdu.l;

  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;


{
 Build DISCONNECT SPDU - category 1
}

PURE PROCEDURE BuildDN(VAR tsdu    : TSDUTYPE;
                       SSuserData : Bytes512TYPE);

VAR SI : ByteTYPE;
    LI : INTEGER;

BEGIN
  tsdu.l := 0;

  SI := 10;
  LI := SSuserData.l;

  BuildHeader(tsdu,SI,LI);

  Build193PGIU(tsdu,SSuserData);
END;
```

```
{
 Build REFUSE SPDU - category 1
}

FURE PROCEDURE BuildRF(VAR tsdu          : TSDUTYPE;
                       CalledSSuserRef : Bytes64TYPE;
                       CommonRef       : Bytes64TYPE;
                       AdditionalRef   : Bytes4TYPE;
                       TCdis           : TCdisTYPE;
                       Srequirements   : FUsetTYPE;
                       VersionNumber   : ByteTYPE;
                       ReasonCode      : ReasonCodeTYPE);

VAR SI    : ByteTYPE;
    LI    : INTEGER;
    spdu : TSDUTYPE;

BEGIN
  tsdu.l := 0;
  spdu.l := 0;

  BuildlbPGIU(spdu,
              CalledSSuserRef,
              CommonSSuserRef,
              AdditionalRef);

  Build17PIU(spdu,TCdis);

  Build20PIU(spdu,Srequirements);

  Build22PIU(spdu,VersionNumber);

  Build50PIU(spdu,ReasonCode);

  SI := 12;
  LI := spdu.l;

  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;
```

```
{
 Build CONNECT SPDU - category 1
}

PURE PROCEDURE BuildCN(VAR tsdu           : TSDUTYPE;
                       CallingSSuserRef : Bytes64TYPE;
                       CommonRef        : Bytes64TYPE;
                       AdditionalRef    : Bytes4TYPE;
                       ProtocolOptions  : ByteTYPE;
                       maxTSDUlen0      : INTEGER;
                       maxTSDUlen1      : INTEGER;
                       VersionNumber    : ByteTYPE;
                       InitialSpsn      : INTEGER;
                       TokenSettingItem : InitialTokensTYPE;
                       Srequirements    : FUsetTYPE;
                       CallingSSAPid    : Bytes16TYPE;
                       CalledSSAPid     : Bytes16TYPE;
                       SSuserData       : Bytes512TYPE);

VAR SI   : ByteTYPE;
    LI   : INTEGER;
    spdu : TSDUTYPE;

BEGIN
  tsdu.l := 0;
  spdu.l := 0;

  BuildlaPGIU(spdu,
              CallingSSuserRef,
              CommonSSuserRef,
              AdditionalRef);

  Build5PGIU(spdu,
             ProtocolOptions,
             maxTSDUlen0,
             maxTSDUlen1,
             VersionNumber,
             InitialSpsn,
             TokenSettingItem);

  Build20PIU(spdu,Srequirements);

  Build51PIU(spdu,CallingSSAPid);

  Build52PIU(spdu,CalledSSAPid);

  Build193PGIU(spdu,SSuserData);

  SI := 13;
  LI := spdu.l;

  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;
```

```
{
 Build ACCEPT SPDU - category 1
}

PURE PROCEDURE BuildAC(VAR tsdu           : TSDUTYPE;
                       CalledSSuserRef    : Bytes64TYPE;
                       CommonRef          : Bytes64TYPE;
                       AdditionalRef      : Bytes4TYPE;
                       ProtocolOptions    : ByteTYPE;
                       maxTSDUlen0        : INTEGER;
                       maxTSDUlen1        : INTEGER;
                       VersionNumber      : ByteTYPE;
                       InitialSpsn        : INTEGER;
                       TokenSettingItem   : InitialTokensTYPE;
                       TokenItem          : TokenSetTYPE;
                       Srequirements      : FUsetTYPE;
                       CallingSSAPid      : Bytes16TYPE;
                       CalledSSAPid       : Bytes16TYPE;
                       SSuserData         : Bytes512TYPE);

VAR SI   : ByteTYPE;
    LI   : INTEGER;
    spdu : TSDUTYPE;

BEGIN
  tsdu.l := 0;
  spdu.l := 0;

  Build1bPGIU(spdu,
              CalledSSuserRef,
              CommonSSuserRef,
              AdditionalRef);

  Build5PGIU(spdu,
             ProtocolOptions,
             maxTSDUlen0,
             maxTSDUlen1,
             VersionNumber,
             InitialSpsn,
             TokenSettingItem);

  Build16PIU(spdu,TokenItem);

  Build20PIU(spdu,Srequirements);

  Build51PIU(spdu,CallingSSAPid);

  Build52PIU(spdu,CalledSSAPid);

  Build193PGIU(spdu,SSuserData);

  SI := 14;
  LI := spdu.l;
  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;
```

```
{
 Build GIVE TOKENS CONFIRM SPDU - category 1
}

PURE PROCEDURE BuildGTC(VAR tsdu : TSDUTYPE);

VAR SI : ByteTYPE;
    LI : INTEGER;

BEGIN
  SI := 21;
  LI := 0;
  BuildHeader(tsdu,SI,LI);
END;


{
 Build GIVE TOKENS ACK SPDU - category 1
}

PURE PROCEDURE BuildGTA(VAR tsdu : TSDUTYPE);

VAR SI : ByteTYPE;
    LI : INTEGER;

BEGIN
  tsdu.l := 0;

  SI := 22;
  LI := 0;

  BuildHeader(tsdu,SI,LI);
END;
```

```
{
 Build ABORT SPDU - category 1
}

PURE PROCEDURE BuildAB(VAR tsdu           : TSDUTYPE;
                       TCdis              : TCdisTYPE;
                       ReflectParameters  : Bytes9TYPE;
                       SSuserData         : Bytes9TYPE);

VAR SI     : ByteTYPE;
    LI,i   : INTEGER;
    spdu   : TSDUTYPE;
    ABdata : Bytes512TYPE;

BEGIN
  tsdu.l   := 0;
  spdu.l   := 0;
  ABdata.l := 0;

  {Convert SSuserData (Bytes9TYPE) to ABdata (Bytes512TYPE)}
  FOR i := 1 TO SSuserData.l DO
    ABdata.d[i] := SSuserData.d[i];
  ABdata.l := SSuserData.l;

  Build17PIU(spdu,TCdis);

  Build49PIU(spdu,ReflectParameters);

  Build193PGIU(spdu,ABdata);

  SI := 25;
  LI := spdu.l;

  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;
```

```
{
 Build ACTIVITY INTERRUPT SPDU - category 2
}

PURE PROCEDURE BuildAI(VAR tsdu   : TSDUTYPE;
                       ReasonCode : ReasonCodeTYPE);

VAR SI   : ByteTYPE;
    LI   : INTEGER;
    spdu : TSDUTYPE;

BEGIN
  tsdu.l := 0;
  spdu.l := 0;

  Build50PIU(spdu,ReasonCode);

  SI := 25;
  LI := spdu.l;

  BuildHeader(tsdu,1,0);   {GT SPDU}
  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;


{
 Build ABORT ACCEPT SPDU - category 1
}

PURE PROCEDURE BuildAA(VAR tsdu : TSDUTYPE);

VAR SI : ByteTYPE;
    LI : INTEGER;

BEGIN
  tsdu.l := 0;

  SI := 26;
  LI := 0;

  BuildHeader(tsdu,SI,LI);
END;
```

```
{
 Build ACTIVITY INTERRUPT ACK SPDU - category 2
}

PURE PROCEDURE BuildAIA(VAR tsdu : TSDUTYPE);

VAR SI : ByteTYPE;
    LI : INTEGER;

BEGIN
  tsdu.l := 0;

  SI := 26;
  LI := 0;

  BuildHeader(tsdu,2,0);   {PT SPDU}
  BuildHeader(tsdu,SI,LI);
END;
```

```
{
 Build ACTIVITY RESUME SPDU - category 2
}

PURE PROCEDURE BuildAR(VAR tsdu          : TSDUTYPE;
                       CalledSSuserRef   : Bytes64TYPE;
                       CallingSSuserRef  : Bytes64TYPE;
                       CommonRef         : Bytes64TYPE;
                       AdditionalRef     : Bytes4TYPE;
                       OldActivityId     : Bytes6TYPE;
                       spsn              : INTEGER;
                       NewActivityId     : Bytes6TYPE;
                       SSuserData        : Bytes512TYPE);

VAR SI   : ByteTYPE;
    LI   : INTEGER;
    spdu : TSDUTYPE;

BEGIN
  tsdu.l := 0;
  spdu.l := 0;

  Build33PIU(spdu,
             CalledSSuserRef,
             CallingSSuserRef,
             CommonSSuserRef,
             AdditionalRef,
             OldActivityId,
             spsn);

  Build193PGIU(spdu,SSuserData);

  SI := 29;
  LI := spdu.l;

  BuildHeader(tsdu,1,0);   {GT SPDU}
  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;
```

```
{
 Build ACTIVITY END SPDU - category 2
}

PURE PROCEDURE BuildAE(VAR tsdu    : TSDUTYPE;
                       spsn        : INTEGER;
                       SSuserData : Bytes512TYPE);

VAR SI   : ByteTYPE;
    LI   : INTEGER;
    spdu : TSDUTYPE;

BEGIN
  tsdu.l := 0;
  spdu.l := 0;

  Build42PIU(spdu,spsn);

  Build193PGIU(spdu,SSuserData);

  SI := 41;
  LI := spdu.l;

  BuildHeader(tsdu,1,0);   {GT SPDU}
  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;


{
 Build ACTIVITY END ACK SPDU - category 2
}

PURE PROCEDURE BuildAEA(VAR tsdu    : TSDUTYPE;
                        spsn        : INTEGER;
                        SSuserData : Bytes512TYPE);

VAR SI   : ByteTYPE;
    LI   : INTEGER;
    spdu : SPDUTYPE;

BEGIN
  tsdu.l := 0;
  spdu.l := 0;

  Build42PIU(spdu,spsn);

  Build193PGIU(spdu,SSuserData);

  SI := 42;
  LI := spdu.l;

  BuildHeader(tsdu,2,0);   {PT SPDU}
  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;
```

```
{
 Build ACTIVITY START SPDU - category 2
}

PURE PROCEDURE BuildAS(VAR tsdu    : TSDUTYPE;
                       ActivityId : Bytes6TYPE;
                       SSuserData : Bytes512TYPE);

VAR SI    : ByteTYPE;
    LI    : INTEGER;
    spdu : TSDUTYPE;

BEGIN
  tsdu.l := 0;
  spdu.l := 0;

  Build41PIU(spdu,ActivityId);

  Build193PGIU(spdu,SSuserData);

  SI := 45;
  LI := spdu.l;

  BuildHeader(tsdu,1,0);   {GT SPDU}
  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;


{
 Build EXCEPTION DATA SPDU - category 2
}

PURE PROCEDURE BuildED(VAR tsdu    : TSDUTYPE;
                       ReasonCode : ReasonCodeTYPE;
                       SSuserData : Bytes512TYPE);

VAR SI    : ByteTYPE;
    LI    : INTEGER;
    spdu : SPDUTYPE;

BEGIN
  tsdu.l := 0;
  spdu.l := 0;

  Build50PIU(spdu,ReasonCode);

  Build193PGIU(spdu,SSuserData);

  SI := 48;
  LI := spdu.l;

  BuildHeader(tsdu,2,0);   {PT SPDU}
  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;
```

```
{
 Build MINOR SYNC POINT SPDU - category 2
}

PURE PROCEDURE BuildMIP(VAR tsdu      : TSDUTYPE;
                        SyncTypeItem : SyncTypeTYPE;
                        spsn         : INTEGER;
                        SSuserData   : Bytes512TYPE);

VAR SI   : ByteTYPE;
    LI   : INTEGER;
    spdu : TSDUTYPE;

BEGIN
  tsdu.1 := 0;
  spdu.1 := 0;

  Build15PIU(spdu,SyncTypeItem);

  Build42PIU(spdu,spsn);

  Build193PGIU(spdu,SSuserData);

  SI := 49;
  LI := spdu.1;

  BuildHeader(tsdu,1,0);   {GT SPDU}
  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;
```

```
{
 Build MINOR SYNC ACK SPDU - category 2
}

PURE PROCEDURE BuildMIA(VAR tsdu    : TSDUTYPE;
                        spsn        : INTEGER;
                        SSuserData : Bytes512TYPE);

VAR SI    : ByteTYPE;
    LI    : INTEGER;
    spdu : TSDUTYPE;

BEGIN
  tsdu.l := 0;
  spdu.l := 0;

  Build42PIU(spdu,spsn);

  Build46PIU(spdu,SSuserData);

  SI := 50;
  LI := spdu.l;

  BuildHeader(tsdu,2,0);    {PT SPDU}
  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;
```

```
{
 Build ACTIVITY DISCARD SPDU - category 2
}

PURE PROCEDURE BuildAD(VAR tsdu    : TSDUTYPE;
                       ReasonCode : ReasonCodeTYPE);

VAR SI   : ByteTYPE;
    LI   : INTEGER;
    spdu : TSDUTYPE;

BEGIN
  tsdu.l := 0;
  spdu.l := 0;

  Build50PIU(spdu,ReasonCode);

  SI := 57;
  LI := spdu.l;

  BuildHeader(tsdu,1,0);   {GT SPDU}
  BuildHeader(tsdu,SI,LI);
  AppendTSDU(tsdu,spdu);
END;


{
 Build ACTIVITY DISCARD ACK SPDU - category 2
}

PURE PROCEDURE BuildADA(VAR tsdu : TSDUTYPE);

VAR SI : ByteTYPE;
    LI : INTEGER;

BEGIN
  tsdu.l := 0;

  SI := 58;
  LI := 0;

  BuildHeader(tsdu,2,0);   {PT SPDU}
  BuildHeader(tsdu,SI,LI);
END;
```

```
{
 PROCEDURE: PIU stripping procedures

            These procedures each strip a unique PIU and its
            parameter value, as part of a SPDU, from a given
            TSDU. Since the inclusion of certain PIUs in a SPDU
            is non-mandatory, these procedures assign appropriate
            default values to required PIU parameters not present
            in the SPDU.

            These procedures only strip those PIUs forming part
            of those SPDUs required by X.400.

            These procedures perform no SPDU structure error
            checking.

 INPUTS:    tsdu      - the TSDU. Its .l field indicates its
                        current length and its .i field points to
                        the last byte stripped.
            parameter - the variable to hold the stripped PIU
                        parameter value.

 OUTPUTS:   parameter - holds the stripped PIU parameter value.
            tsdu.i    - is updated to exclude the stripped PIU,
                        if it was present.

 CALLS:     StripHeader, bitAND.
}
```

```
{
 Strip Called SS-user reference
}

PURE PROCEDURE Strip9PIU(VAR tsdu             : TSDUTYPE;
                         VAR CalledSSuserRef : Bytes64TYPE);

VAR LI,i : INTEGER;

BEGIN
  CalledSSuserRef.l := 0;

  StripHeader(tsdu,9,LI);
  IF LI > 0
  THEN
    BEGIN
      FOR i := 1 TO LI DO
         CalledSSuserRef.d[i] := tsdu.d[tsdu.i+i];
      CalledSSuserRef.l := LI;
      tsdu.i := tsdu.i + LI;
    END;
END;


{
 Strip Calling SS-user reference
}

PURE PROCEDURE Strip10PIU(VAR tsdu             : TSDUTYPE;
                          VAR CallingSSuserRef : Bytes64TYPE);

VAR LI,i : INTEGER;

BEGIN
  CallingSSuserRef.l := 0;

  StripHeader(tsdu,10,LI);
  IF LI > 0
  THEN
    BEGIN
      FOR i := 1 TO LI DO
        CallingSSuserRef.d[i] := tsdu.d[tsdu.i+i];
      CallingSSuser.l := LI;
      tsdu.i := tsdu.i + LI;
    END;
END;
```

```
{
 Strip Common reference
}

PURE PROCEDURE Strip11PIU(VAR tsdu      : TSDUTYPE;
                          VAR CommonRef : Bytes64TYPE);

VAR LI,i : INTEGER;

BEGIN
  CommonRef.l := 0;

  StripHeader(tsdu,11,LI);
  IF LI > 0
  THEN
    BEGIN
      FOR i := 1 TO LI DO
        CommonRef.d[i] := tsdu.d[tsdu.i+i];
      CommonRef.l := LI;
      tsdu.i := tsdu.i + LI;
    END;
END;


{
 Strip Additional reference information
}

PURE PROCEDURE Strip12PIU(VAR tsdu          : TSDUTYPE;
                          VAR AdditionalRef : Bytes4TYPE);

VAR LI,i : INTEGER;

BEGIN
  AdditionalRef.l := 0;

  StripHeader(tsdu,12,LI);
  IF LI > 0
  THEN
    BEGIN
      FOR i := 1 TO LI DO
        AdditionalRef.d[i] := tsdu.d[tsdu.i+i];
      AdditionalRef.l := LI;
      tsdu.i := tsdu.i + LI;
    END;
END;
```

```
{
 Strip Sync type item
}

PURE PROCEDURE Stripl5PIU(VAR tsdu          : TSDUTYPE;
                          VAR SyncTypeItem : SyncTypeTYPE);

VAR LI : INTEGER;

BEGIN
  SyncTypeItem := EXPLICIT;

  StripHeader(tsdu,15,LI);
  IF LI > 0
  THEN
    BEGIN
      IF bitAND(tsdu.d[tsdu.i+1],1) = 1
      THEN
        SyncTypeItem := OPTIONAL;
      tsdu.i := tsdu.i + LI;
    END;
END;
```

```
{
 Strip Token item
}

PURE PROCEDURE Strip16PIU(VAR tsdu      : TSDUTYPE;
                         VAR TokenItem : TokenSetTYPE);

VAR LI,mask : INTEGER;
    token   : TokenTYPE;
    byte    : ByteTYPE;

BEGIN
  TokenItem := [];

  StripHeader(tsdu,16,LI);
  IF LI > 0
  THEN
    BEGIN
      TokenItem := [];
      byte      := tsdu.d[tsdu.i+1];
      mask      := 64;

      FOR token := TRT DOWNTO DKT DO
        BEGIN
          IF bitAND(mask,byte) = 1
          THEN
            TokenItem := TokenItem + [token];
          mask := mask DIV 4;
        END;

      tsdu.i := tsdu.i + LI;
    END;
END;
```

```
{
 Strip Transport disconnect
{

PURE PROCEDURE Stripl7PIU(VAR tsdu  : TSDUTYPE;
                          VAR TCdis : TCdisTYPE);

VAR LI   : INTEGER;
    byte : ByteTYPE;

BEGIN
  TCdis.TCkept   := FALSE;
  TCdis.ABreason := NO_ABORT;

  StripHeader(tsdu,17,LI);
  IF LI > 0
  THEN
    BEGIN
      byte := tsdu.d[tsdu.i+1];
      WITH TCdis DO
        BEGIN
          TCkept := (bitAND(byte,1) = 0);

          CASE bitAND(byte,14) OF
            0 : ABreason := NO_ABORT;
            2 : ABreason := USER_ABORT;
            4 : ABreason := PROTOCOL_ERROR;
            8 : ABreason := NO_REASON;
          END;

        END;
      tsdu.i := tsdu.i + LI;
    END;
END;


{
 Strip Protocol options
}

PURE PROCEDURE Stripl9PIU(VAR tsdu            : TSDUTYPE;
                          VAR ProtocolOptions : ByteType);

VAR LI : INTEGER;

BEGIN
  ProtocolOptions := 0;

  StripHeader(tsdu,19,LI);
  IF LI > 0
  THEN
    BEGIN
      ProtocolOptions := tsdu.d[tsdu.i+1];
      tsdu.i := tsdu.i + LI;
    END;
END;
```

```
{
 Strip Session user requirements
}

PURE PROCEDURE Strip20PIU(VAR tsdu          : TSDUTYPE;
                         VAR Srequirements : FUsetTYPE);

VAR LI,mask : INTEGER;
    fu      : FUTYPE;
    byte    : ByteTYPE;

BEGIN
  Srequirements := FU_SUP;

  StripHeader(tsdu,20,LI);
  IF LI > 0
  THEN
    BEGIN
      Srequirements := [];
      mask := 128;
      byte := tsdu.d[tsdu.i+2];

      FOR fu := NR DOWNTO HD DO
        BEGIN
          IF bitAND(mask,byte) = 1
          THEN
            Srequirements := Srequirements + [fu];
          mask := mask DIV 2;
        END;

      mask := 4;
      byte := tsdu.d[tsdu.i+1];

      FOR fu := TD DOWNTO CD DO
        BEGIN
          IF bitAND(mask,byte) = 1
          THEN
            Srequirements := Srequirements + [fu];
          mask := mask DIV 2;
        END;

      tsdu.i := tsdu.i + LI;
    END;
END;
```

```
{
 Strip TSDU maximum size
}

PURE PROCEDURE Strip21PIU(VAR tsdu         : TSDUTYPE;
                          VAR maxTSDUlen0 : INTEGER;
                          VAR maxTSDUlen1 : INTEGER);

VAR LI : INTEGER;

BEGIN
  maxTSDUlen0 := 0;
  maxTSDUlen1 := 0;

  StripHeader(tsdu,21,LI);
  IF LI > 0
  THEN
    BEGIN
      maxTSDUlen0 := ORD(tsdu[tsdu.i+1])*256 +
                     ORD(tsdu[tsdu.i+2]);
      maxTSDUlen1 := ORD(tsdu[tsdu.i+3])*256 +
                     ORD(tsdu[tsdu.i+4]);
      tsdu.i := tsdu.i + LI;
    END;
END;


{
 Strip Version number
}

PURE PROCEDURE Strip22PIU(VAR tsdu          : TSDUTYPE;
                          VAR VersionNumber : ByteTYPE);

VAR LI : INTEGER;

BEGIN
  VersionNumber := VERSION;

  StripHeader(tsdu,22,LI);
  IF LI > 0
  THEN
    BEGIN
      VersionNumber := tsdu[tsdu.i+1];
      tsdu.i := tsdu.i + LI;
    END;
END;
```

```
{
 Strip Initial serial number
}

PURE PROCEDURE Strip23PIU(VAR tsdu        : TSDUTYPE;
                         VAR InitialSpsn : INTEGER);

VAR LI,i,factor : INTEGER;

BEGIN
  InitialSpsn := 0;

  StripHeader(tsdu,23,LI);
  IF LI >0
  THEN
    BEGIN
      InitialSpsn := 0;
      factor := 1;

      FOR i := LI DOWNTO 1 DO
        BEGIN
          InitialSpsn := InitialSpsn +
                         (ORD(tsdu[tsdu.i+i]-48)*factor;
          factor := factor * 10;
        END;

      tsdu.i := tsdu.i + LI;
    END;
END;
```

```
{
 Strip Enclosure item
}

PURE PROCEDURE Strip25PIU(VAR tsdu          : TSDUTYPE;
                         VAR EnclosureItem : EnclosureItemTYPE);

VAR LI : INTEGER;

BEGIN
  EnclosureItem := BEGIN_END;

  StripHeader(tsdu,25,LI);
  IF LI > 0
  THEN
    BEGIN

      CASE BitAND(tsdu[tsdu.i+1],3) OF
        0 : EnclosureItem := NOT_BEGIN_NOT_END;
        1 : EnclosureItem := BEGIN_NOT_END;
        2 : EnclosureItem := NOT_BEGIN_END;
        3 : EnclosureItem := BEGIN_END;
      END;

      tsdu.i := tsdu.i + LI;
    END;
END;
```

```
{
 Strip Token setting item
}

PURE PROCEDURE Strip26PIU(VAR tsdu          : TSDUTYPE;
                         VAR TokenSetItem : InitialTokensTYPE);

VAR LI,mask    : INTEGER;
    token      : TokenTYPE;
    bits,byte  : ByteTYPE;

BEGIN
  TokenSetItem := DEFAULT_TKNS;

  StripHeader(tsdu,26,LI);
  IF LI > 0
  THEN
    BEGIN
      byte   := tsdu.d[tsdu.i+1];
      mask   := 3;
      shiftR := 1;

      FOR token := DKT TO TRT DO
        BEGIN
          bits := ORD(bitAND(mask,byte)) DIV shiftR;

          CASE bits OF
            0 : TokenSetItem[token] := REQUESTOR_SIDE;
            1 : TokenSetItem[token] := ACCEPTOR_SIDE;
            2 : TokenSetItem[token] := ACCEPTOR_CHOOSES;
            3 : ; {Reserved}
          END;

          mask   := mask   * 4;
          shiftR := shiftR * 4;
        END;

      tsdu.i := tsdu.i + LI;
    END;
END;
```

```
{
 Strip Activity identifier
}

PURE PROCEDURE Strip41PIU(VAR tsdu       : TSDUTYPE;
                         VAR ActivityId : Bytes6TYPE);

VAR LI,i : INTEGER;

BEGIN
  ActivityId.l := 0;

  StripHeader(tsdu,41,LI);
  IF LI > 0
  THEN
    BEGIN
      FOR I := 1 TO LI DO
        ActivityId.d[i] := tsdu.d[tsdu.i+i];
      ActivityId.l := LI;
      tsdu.i := tsdu.i + LI;
    END;
END;


{
 Strip Serial number
}

PURE PROCEDURE Strip42PIU(VAR tsdu : TSDUTYPE;
                          VAR spsn : INTEGER);

VAR LI,i,factor : INTEGER;

BEGIN
  spsn := 0;

  StripHeader(tsdu,42,LI);
  IF LI > 0
  THEN
    BEGIN
      spsn   := 0;
      factor := 1;

      FOR i := LI DOWNTO 1 DO
        BEGIN
          spsn := spsn + (ORD(tsdu.d[tsdu.i+i])-48)*factor;
          factor := factor * 10;
        END;

      tsdu.i := tsdu.i + LI;
    END;
END;
```

```
{
 Strip User data
}

PURE PROCEDURE Strip46PIU(VAR tsdu        : TSDUTYPE;
                          VAR SSuserData : Bytes512TYPE);

VAR L,i: INTEGER;

BEGIN
  SSuserData.l := 0;

  StripHeader(tsdu,46,LI);
  IF LI > 0
  THEN
    BEGIN
      FOR I := 1 TO LI DO
        SSuserData.d[i] := tsdu.d[tsdu.i+i];
      SSuserData.l := LI;
      tsdu.i := tsdu.i + LI;
    END;
END;


{
 Strip Reflect parameter values
}

PURE PROCEDURE Strip49PIU(VAR tsdu              : TSDUTYPE;
                          VAR ReflectParameters : Bytes9TYPE);

VAR LI,i : INTEGER;

BEGIN
  ReflectParameters.l := 0;

  StripHeader(tsdu,49,LI);
  IF LI > 0
  THEN
    BEGIN
      FOR i := 1 TO LI DO
        ReflectParameters.d[i] := tsdu.d[tsdu.i+i];
      ReflectParameters.l := LI;
      tsdu.i := tsdu.i + LI;
    END;
END;
```

```
{
 Strip Reason code
}

PURE PROCEDURE Strip50PIU(VAR tsdu       : TSDUTYPE;
                         VAR ReasonCode : ReasonCodeTYPE);

VAR LI,i,InfoLen : INTEGER;

BEGIN
  ReasonCode.Data.l := 0;
  ReasonCode.Reason := SSU_UNSPECIFIED;

  StripHeader(tsdu,50,LI);
  IF LI > 0
  THEN
    BEGIN
      WITH ReasonCode DO
        BEGIN

          CASE bitAND(tsdu.d(tsdu.i+1),136) OF
             0 : Reason := SSU_UNSPECIFIED;
             1 : Reason := SSU_CONGESTED;
             2 : Reason := SSU_SEE_DATA;
             3 : Reason := SEQUENCE_ERROR;
             5 : Reason := LOCAL_SSU_ERROR;
             6 : Reason := PROCEDURE_ERROR;
           128 : Reason := DEMAND_DK;
           129 : Reason := CALLED_SSAP_UNKNOWN;
           130 : Reason := CALLED_SSU_UNATTACHED;
           131 : Reason := SSP_CONGESTED;
           132 : Reason := PROPOSED_PROTOCOL;
           4,7,8,133,134,135,136 : ; {Reserved}
          END;

          InfoLen := LI - 1;
          IF InfoLen > 0
          THEN
            FOR i := 1 TO InfoLen DO
              Data.d[i] := tsdu.d[tsdu.i+1+i];
          Data.l := InfoLen;
        END;
      tsdu.i := tsdu.i + LI;
    END;
END;
```

```
{
 Strip Calling SSAP identifier
}

PURE PROCEDURE Strip51PIU(VAR tsdu          : TSDUTYPE;
                         VAR CallingSSAPid : Bytes16TYPE);

VAR LI,i : INTEGER;

BEGIN
  CallingSSAPid.l := 0;

  StripHeader(tsdu,51,LI);
  IF LI > 0
  THEN
    BEGIN
      FOR i := 1 TO LI DO
        CallingSSAPid.d[i] := tsdu.d[tsdu.i+i];
      CallingSSAPid.l := LI;
      tsdu.i := tsdu.i + LI;
     END;
END;


{
 Strip Called SSAP identifier
}

PURE PROCEDURE Strip52PIU(VAR tsdu          : TSDUTYPE;
                         VAR CalledSSAPid : Bytes16TYPE);

VAR LI,i : INTEGER;

BEGIN
  CalledSSAPid.l := 0;

  StripHeader(tsdu,52,LI);
  IF LI > 0
  THEN
    BEGIN
      FOR i := 1 TO LI DO
        CalledSSAPid.d[i] := tsdu.d[tsdu.i+i];
      CalledSSAPid.l := LI;
      tsdu.i := tsdu.i + LI;
     END;
END;
```

```
{
 PROCEDURE: PGIU stripping procedures

            These procedures each strip a unique PGIU and its
            parameter values, as part of a SPDU, from a given
            TSDU.

            These procedures only strip those PGIUs forming part
            of those SPDUs required by X.400.

            These procedures perform no SPDU structure error
            checking.

 INPUTS:    tsdu       - the TSDU. Its .l field indicates its
                         current length and its .i field points
                         to the last byte stripped.
            parameters - the variables to hold the stripped PGIU
                         parameter values.

 OUTPUTS:   parameters - hold the stripped PGIU parameter values.
            tsdu.i     - is updated to exclude the stripped PGIU,
                         if it was present.

 CALLS:     StripHeader, PIU stripping procedures.
}
```

```
{
 Strip Connection identifier for CN SPDU
}

PURE PROCEDURE StriplaPGIU(VAR tsdu              : TSDUTYPE;
                           VAR CallingSSuserRef  : Bytes64TYPE;
                           VAR CommonRef         : Bytes64TYPE;
                           VAR AdditionalRef     : Bytes4TYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,1,LI);

  Stripl0PIU(tsdu,CallingSSuserRef);

  Stripl1PIU(tsdu,CommonRef);

  Stripl2PIU(tsdu,AdditionalRef);
END;

{
 Strip Connection identifier for AC, RF SPDUs
}

PURE PROCEDURE StriplbPGIU(VAR tsdu              : TSDUTYPE;
                           VAR CalledSSuserRef   : Bytes64TYPE;
                           VAR CommonRef         : Bytes64TYPE;
                           VAR AdditionalRef     : Bytes4TYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,1,LI);

  Strip9PIU(tsdu,CalledSSuserRef);

  Stripl1PIU(tsdu,CommonRef);

  Stripl2PIU(tsdu,AdditionalRef);
END;
```

```
{
 Strip Connect/Accept item
}

PURE PROCEDURE Strip5PGIU(VAR tsdu             : TSDUTYPE;
                         VAR ProtocolOptions : ByteTYPE;
                         VAR maxTSDUlen0     : INTEGER;
                         VAR maxTSDUlen1     : INTEGER;
                         VAR VersionNumber   : ByteTYPE;
                         VAR InitialSpsn     : INTEGER;
                         VAR TokenSetItem    : InitialTokensTYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,5,LI);

  Strip19PIU(tsdu,ProtocolOptions);

  Strip21PIU(tsdu,maxTSDUlen0,maxTSDUlen1);

  Strip22PIU(tsdu,VersionNumber);

  Strip23PIU(tsdu,InitialSpsn);

  Strip26PIU(tsdu,TokenSetItem);
END;
```

```
{
 Strip Linking information
}

PURE PROCEDURE Strip33PGIU(VAR tsdu              : TSDUTYPE;
                           VAR CalledSSuserRef   : Bytes64TYPE;
                           VAR CallingSSuserRef  : Bytes64TYPE;
                           VAR CommonRef         : Bytes64TYPE;
                           VAR AdditionalRef     : Bytes4TYPE;
                           VAR OldActivityId     : Bytes6TYPE;
                           VAR spsn              : INTEGER);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,33,LI);

  Strip9PIU(tsdu,CalledSSuserRef);

  Strip10PIU(tsdu,CallingSSuserRef);

  Strip11PIU(tsdu,CommonRef);

  Strip12PIU(tsdu,AdditionalRef);

  Strip41PIU(tsdu,OldActivityId);

  Strip42PIU(tsdu,spsn);
END;
```

```
{
 Strip User data
}

PURE PROCEDURE Stripl93PGIU(VAR tsdu       : TSDUTYPE;
                            VAR SSuserData : Bytes512TYPE);

VAR LI,i : INTEGER;

BEGIN
  SSuserData.l := 0;

  StripHeader(tsdu,193,LI);
  IF LI > 0
  THEN
    BEGIN
      FOR i := 1 TO LI DO
        SSuserData.d[i] := tsdu.d[tsdu.i+i];
      SSuserData.l := LI;
      tsdu.i := tsdu.i + LI;
    END;
END;


{
 Strip User information field
}

PURE PROCEDURE StripUserInfo(VAR tsdu     : TSDUTYPE;
                             VAR UserInfo : SSDUTYPE);

VAR LI,i,InfoLen : INTEGER;

BEGIN
  UserInfo.l := 0;

  InfoLen := tsdu.l - tsdu.i;
  IF InfoLen > 0
  THEN
    BEGIN
      FOR i := 1 TO InfoLen DO
        UserInfo.d[i] := tsdu.d[tsdu.i+i];
      UserInfo.l := InfoLen;
      tsdu.i := tsdu.i + InfoLen;
    END;
END;
```

```
{
PROCEDURE: SPDU stripping procedures

           These procedures each strip a unique SPDU and its
           parameter values from a given TSDU.

           These procedures only strip those SPDUs required by
           X.400, and of these only those with at least one
           parameter field.

           These procedures perform no SPDU structure error
           checking.

           Category 0 & 1 SPDUs are stripped starting from the
           1st TSDU byte. Category 2 SPDUs are preceded by
           either a GT or PT SPDU according to the rules of
           Basic Concatenation. Since the X.400 SPM makes no real
           use of this feature, such GT and PT SPDUs have no
           parameter fields and are therefore reduced to their
           headers only. They are ignored by these procedures.

INPUTS:    tsdu       - the TSDU. Its .l field indicates its
                        length and its .i field = 0
                        (start of TSDU).

           parameters - the variables to hold the stripped SPDU
                        parameter values.

OUTPUTS:   parameters - hold the stripped SPDU parameter values.

CALLS:     StripHeader,
           PIU and PGIU stripping procedures.
}
```

```
{
 Strip DATA TRANSFER SPDU - category 2
}

PURE PROCEDURE StripDT(VAR tsdu          : TSDUTYPE;
                       VAR EnclosureItem : EnclosureItemTYPE;
                       VAR UserInfo      : SSDUTYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,1,LI); {GT SPDU}
  StripHeader(tsdu,1,LI);

  Strip25PIU(tsdu,EnclosureItem);

  StripUserInfo(tsdu,UserInfo);
END;


{
 Strip PLEASE TOKENS SPDU - category 0
}

PURE PROCEDURE StripPT(VAR tsdu      : TSDUTYPE;
                       VAR TokenItem : TokenSetTYPE;
                       VAR SSuserData : Bytes512TYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,2,LI);

  Strip16PIU(tsdu,TokenItem);

  Strip193PGIU(tsdu,SSuserData);
END;
```

```
{
 Strip FINISH SPDU - category 1
}

PURE PROCEDURE StripFN (VAR tsdu       : TSDUTYPE;
                        VAR TCdis      : TCdisTYPE;
                        VAR SSuserData : Bytes512TYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,9,LI);

  Strip17PIU(tsdu,TCdis);

  Strip193PGIU(tsdu,SSuserData);
END;


{
 Strip DISCONNECT SPDU - category 1
}

PURE PROCEDURE StripDN(VAR tsdu       : TSDUTYPE;
                       VAR SSuserData : Bytes512TYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,10,LI);

  Strip193PGIU(tsdu,SSuserData);
END;
```

```
{
 Strip REFUSE SPDU - category 1
}

PURE PROCEDURE StripRF(VAR tsdu             : TSDUTYPE;
                       VAR CalledSSuserRef  : Bytes64TYPE;
                       VAR CommonRef        : Bytes64TYPE;
                       VAR AdditionalRef    : Bytes4TYPE;
                       VAR TCdis            : TCdisTYPE;
                       VAR Srequirements    : FUsetTYPE;
                       VAR VersionNumber    : ByteTYPE;
                       VAR ReasonCode       : ReasonCodeTYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,12,LI);

  Strip1bPGIU(tsdu,
              CalledSSuserRef,
              CommonRef,
              AdditionalRef);

  Strip17PIU(tsdu,TCdis);

  Strip20PIU(tsdu,Srequirements);

  Strip22PIU(tsdu,VersionNumber);

  Strip50PIU(tsdu,ReasonCode);
END;
```

```
{
 Strip CONNECT SPDU - category 1
}

PURE PROCEDURE StripCN(VAR tsdu              : TSDUTYPE;
                       VAR CallingSSuserRef  : Bytes64TYPE;
                       VAR CommonRef         : Bytes64TYPE;
                       VAR AdditionalRef     : Bytes4TYPE;
                       VAR ProtocolOptions   : ByteTYPE;
                       VAR maxTSDUlen0       : INTEGER;
                       VAR maxTSDUlen1       : INTEGER;
                       VAR VersionNumber     : ByteTYPE;
                       VAR InitialSpsn       : INTEGER;
                       VAR TokenSettingItem  : InitialTokensTYPE;
                       VAR Srequirements     : FUsetTYPE;
                       VAR CallingSSAPid     : Bytes16TYPE;
                       VAR CalledSSAPid      : Bytes16TYPE;
                       VAR SSuserData        : Bytes512TYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,13,LI);

  StriplaPGIU(tsdu,
              CallingSSuserRef,
              CommonRef,
              AdditionalRef);

  Strip5PGIU(tsdu,
             ProtocolOptions,
             maxTSDUlen0,
             maxTSDUlen1,
             VersionNumber,
             InitialSpsn,
             TokenSettingItem);

  Strip20PIU(tsdu,Srequirements);

  Strip51PIU(tsdu,CallingSSAPid);

  Strip52PIU(tsdu,CalledSSAPid);

  Strip193PGIU(tsdu,SSuserData);
END;
```

```
{
 Strip ACCEPT SPDU - category 1
}

PURE PROCEDURE StripAC(VAR tsdu              : TSDUTYPE;
                       VAR CalledSSuserRef   : Bytes64TYPE;
                       VAR CommonRef         : Bytes64TYPE;
                       VAR AdditionalRef     : Bytes4TYPE;
                       VAR ProtocolOptions   : ByteTYPE;
                       VAR maxTSDUlen0       : INTEGER;
                       VAR maxTSDUlen1       : INTEGER;
                       VAR VersionNumber     : ByteTYPE;
                       VAR InitialSpsn       : INTEGER;
                       VAR TokenSettingItem  : InitialTokensTYPE;
                       VAR TokenItem         : TokenSetTYPE;
                       VAR Srequirements     : FUsetTYPE;
                       VAR CallingSSAPid     : Bytes16TYPE;
                       VAR CalledSSAPid      : Bytes16TYPE;
                       VAR SSuserData        : Bytes512TYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,14,LI);

  Strip1bPGIU(tsdu,
              CalledSSuserRef,
              CommonRef,
              AdditionalRef);

  Strip5PGIU(tsdu,
             ProtocolOptions,
             maxTSDUlen0,
             maxTSDUlen1,
             VersionNumber,
             InitialSpsn,
             TokenSettingItem);

  Strip16PIU(tsdu,TokenItem);

  Strip20PIU(tsdu,Srequirements);

  Strip51PIU(tsdu,CallingSSAPid);

  Strip52PIU(tsdu,CalledSSAPid);

  Strip193PGIU(tsdu,SSuserData);
END;
```

```
{
 Strip ABORT SPDU - category 1
}

PURE PROCEDURE StripAB(VAR tsdu              : TSDUTYPE;
                       VAR TCdis             : TCdisTYPE;
                       VAR ReflectParameters : Bytes9TYPE;
                       VAR SSuserData        : Bytes9TYPE);

VAR LI,i   : INTEGER;
    ABdata : Bytes512TYPE;

BEGIN
  StripHeader(tsdu,25,LI);

  Strip17PIU(tsdu,TCdis);

  Strip49PIU(tsdu,ReflectParameters);

  Strip193PGIU(tsdu,ABdata);

  IF ABdata.l > 0
  THEN

    {Convert ABdata (Bytes512TYPE) to SSuserData (Bytes9TYPE)}

    FOR i := 1 TO ABdata.l DO
      SSuserData.d[i] := ABdata.d[i];

  SSuserData.l := ABdata.l;
END;


{
 Strip ACTIVITY INTERRUPT SPDU - category 2
}

PURE PROCEDURE StripAI(VAR tsdu       : TSDUTYPE;
                       VAR ReasonCode : ReasonCodeTYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,1,LI);  {GT SPDU}
  StripHeader(tsdu,25,LI);

  Strip50PIU(tsdu,ReasonCode);
END;
```

```
{
 Strip ACTIVITY RESUME SPDU - category 2
}

PURE PROCEDURE StripAR(VAR tsdu              : TSDUTYPE;
                       VAR CalledSSuserRef   : Bytes64TYPE;
                       VAR CallingSSuserRef  : Bytes64TYPE;
                       VAR CommonRef         : Bytes64TYPE;
                       VAR AdditionalRef     : Bytes4TYPE;
                       VAR OldActivityID     : Bytes6TYPE;
                       VAR spsn              : INTEGER;
                       VAR NewActivityId     : Bytes6TYPE;
                       VAR SSuserData        : Bytes512TYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,1,LI);  {GT SPDU}
  StripHeader(tsdu,29,LI);

  Strip33PGIU(tsdu,
              CalledSSuserRef,
              CallingSSuserRef,
              CommonRef,
              AdditionalRef,
              OldActivityId,
              spsn);

  Strip41PIU(tsdu,NewActivityId);

  Strip193PGIU(tsdu,SSuserData);
END;


{
 Strip ACTIVITY END SPDU - category 2
}

PURE PROCEDURE StripAE(VAR tsdu       : TSDUTYPE;
                       VAR spsn       : INTEGER;
                       VAR SSuserData : Bytes512TYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,1,LI);  {GT SPDU}
  StripHeader(tsdu,41,LI);

  Strip42PIU(tsdu,spsn);

  Strip193PGIU(tsdu,SSuserData);
END;
```

```
{
 Strip ACTIVITY END ACK SPDU - category 2
}

PURE PROCEDURE StripAEA(VAR tsdu       : TSDUTYPE;
                        VAR spsn       : INTEGER;
                        VAR SSuserData : Bytes512TYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,2,LI);  {PT SPDU}
  StripHeader(tsdu,42,LI);

  Strip42PIU(tsdu,spsn);

  Stripl93PGIU(tsdu,SSuserData);
END;


{
 Strip ACTIVITY START SPDU - category 2
}

PURE PROCEDURE StripAS(VAR tsdu       : TSDUTYPE;
                       VAR ActivityId : Bytes6TYPE;
                       VAR SSuserData : Bytes512TYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,1,LI);  {GT SPDU}
  StripHeader(tsdu,45,LI);

  Strip41PIU(tsdu,ActivityId);

  Stripl93PGIU(tsdu,SSuserData);
END;
```

```
{
 Strip EXCEPTION DATA SPDU - category 2
}

PURE PROCEDURE StripED(VAR tsdu       : TSDUTYPE;
                       VAR ReasonCode : ReasonCodeTYPE;
                       VAR SSuserData : Bytes512TYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,2,LI);  {PT SPDU}
  StripHeader(tsdu,48,LI);

  Strip50PIU(tsdu,ReasonCode);

  Strip193PGIU(tsdu,SSuserData);
END;


{
 Strip MINOR SYNC POINT SPDU - category 2
}

PURE PROCEDURE StripMIP(VAR tsdu         : TSDUTYPE;
                        VAR SyncTypeItem : SyncTypeTYPE;
                        VAR spsn         : INTEGER;
                        VAR SSuserData   : Bytes512TYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,1,LI);  {GT SPDU}
  StripHeader(tsdu,49,LI);

  Strip15PIU(tsdu,SyncTypeItem);

  Strip42PIU(tsdu,spsn);

  Strip193PGIU(tsdu,SSuserData);
END;
```

```
{
 Strip MINOR SYNC ACK SPDU - category 2
}

PURE PROCEDURE StripMIA(VAR tsdu        : TSDUTYPE;
                        VAR spsn        : INTEGER;
                        VAR SSuserData : Bytes512TYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,2,LI);  {PT SPDU}
  StripHeader(tsdu,50,LI);

  Strip42PIU(tsdu,spsn);

  Strip46PIU(tsdu,SSuserData);
END;


{
 Strip ACTIVITY DISCARD SPDU - category 2
}

PURE PROCEDURE StripAD(VAR tsdu       : TSDUTYPE;
                       VAR ReasonCode : ReasonCodeTYPE);

VAR LI : INTEGER;

BEGIN
  StripHeader(tsdu,1,LI);  {GT SPDU}
  StripHeader(tsdu,57,LI);

  Strip50PIU(tsdu,ReasonCode);
END;
```

```
{
 FUNCTION: SpsnVal

           This function gets the Serial Number parameter value
           from a MIP, MIA, AE or AEA SPDU in a given TSDU.

           NOTE: MIP, MIA, AE and AEA are category 2 SPDUs, so
                 the first SPDU in the TSDU (GT or PT) is ignored.

 INPUTS:   tsdu - the TSDU containing the
                  MIP, MIA, AE or AEA SPDU.
                  Its .l field indicates its length and its
                  .i field = 0 (start of TSDU).

 OUTPUTS:  returns: the Serial Number parameter value.

 CALLS:    StripHeader, Strip42PIU.
}


PURE FUNCTION SpsnVal(tsdu : TSDUTYPE) : INTEGER;

VAR spsn, LI : INTEGER;

BEGIN
  StripHeader(tsdu,0,LI);  {remove GT or PT SPDU}

  StripHeader(tsdu,0,LI);  {remove SPDU header}

  StripHeader(tsdu,15,LI); {remove PIU 15 header if present}
  tsdu.i := tsdu.i + LI;   {skip   PIU 15 if present}

  Strip42PIU(tsdu,spsn);   {strip Serial number parameter}
  SpsnVal := spsn;
END;
```

```
{
 FUNCTION: reuseTC

           This function determines whether a RF, AB or FN SPDU
           in a given TSDU requests reuse of the transport
           connection, i.e.: whether the SPDU
           is RFr,  ABr   or FNr  (TC reused),
           or RFnr, ABnr, or FNnr (TC not reused).

           NOTE: RF, AB and FN are category 1 SPDUs, so they are
                 the first and only SPDUs in the TSDU.

 INPUTS:   tsdu - the TSDU containing the RF, AB or FN SPDU.
                  Its .l field indicates its length and its
                  .i field = 0 (start of TSDU).

 OUTPUTS:  returns: TRUE  if SPDU is RFr,  ABr  or FNr,
                    FALSE if SPDU is RFnr, ABnr or FNnr.

 CALLS:    StripHeader, Stripl7PIU.
}


PURE FUNCTION reuseTC(tsdu : TSDUTYPE) : BOOLEAN;

VAR LI    : INTEGER;
    TCdis : TCdisTYPE;

BEGIN
  StripHeader(tsdu,0,LI); {remove SPDU header}

  StripHeader(tsdu,1,LI); {remove PGIU 1 header if present}
  tsdu.i := tsdu.i + LI;  {skip   PGIU 1 if present}

  Stripl7PIU(tsdu,TCdis); {strip Transport Disconnect parameter}
  reuseTC := TCdis.TCkept;
END;
```

```
{
 FUNCTION: TokensVal

           This function gets the Token Item parameter value from
           a AC or PT SPDU in a given TSDU.

           NOTE: AC and PT are category 1 and 0 SPDUs
                 respectively, so they will be the first SPDUs
                 in the TSDU.

 INPUTS:   tsdu - the TSDU containing the AC or PT SPDU.
                  Its .l field indicates its length and its
                  .i field = 0 (start of TSDU).

 OUTPUTS:  returns: the Token Item parameter value.

 CALLS:    StripHeader, Stripl6PIU.
}


PURE FUNCTION TokensVal(tsdu : TSDUTYPE) : TokenSetTYPE;

VAR TokenItem : TokenSetTYPE;
    LI        : INTEGER;

BEGIN
  StripHeader(tsdu,0,LI);      {remove SPDU header}

  StripHeader(tsdu,1,LI);      {remove PGIU 1 header if present}
  tsdu.i := tsdu.i + LI;       {skip   PGIU 1 if present}

  StripHeader(tsdu,5,LI);      {remove PGIU 5 header if present}
  tsdu.i := tsdu.i + LI;       {skip   PGIU 5 if present}

  Stripl6PIU(tsdu,TokenItem); {strip Token Item parameter}
  TokensVal := TokenItem;
END;
```

```
{
 FUNCTION: idSPDU

           This function determines whether the SPDU contained in
           a given TSDU is a particular, required one.

           This function ignores the GT or PT SPDUs which always
           precede category 2 SPDUs in the TSDU. Such GT and PT
           SPDUs have no parameter fields. Only the concatenated
           category 2 SPDU is checked.

 INPUTS:   tsdu   - the TSDU. Its .l field indicates its length
                    and its .i field = 0 (start of TSDU).

           SPDUid - specifies the required SPDU.

 OUTPUTS:  returns: TRUE:  if the SPDU is the required one,
                    FALSE: if not.

 CALLS:    reuseTC.
}


PURE FUNCTION idSPDU(tsdu   : TSDUTYPE;
                     SPDUid : SPDUidTYPE) : BOOLEAN;

VAR SI    : ByteTYPE; {SPDU identifier}
    LI    : INTEGER;  {SPDU length indicator}
    Cat01 : BOOLEAN;  {SPDU is category 0 or 1 - 1st in TSDU}
    Cat2  : BOOLEAN;  {SPDU is category 2      - 2nd in TSDU}

BEGIN
  IF tsdu.l > 0
  THEN                {tsdu not empty}
    BEGIN
      Cat01 := FALSE;
      Cat2  := FALSE;
      SI    := tsdu.d[1];
      LI    := tsdu.d[2];

      {
       Check for GT or PT SPDU preceding a category 2 SPDU.
       If one is found, ignore it and get the SI value of the
       category 2 SPDU.
      }

      IF ((SI = 1) OR (SI = 2)) AND (LI = 0)
      THEN     {category 2 SPDU}
        BEGIN
          SI   := tsdu.d[3];
          Cat2 := TRUE;
        END;
      ELSE     {category 0 or 1 SPDU}
        Cat01 := TRUE;
```

```
      CASE SPDUid OF
      DT     : idSPDU := (SI = 1) AND Cat2;
      PT     : idSPDU := (SI = 2);

      FN     : idSPDU := (SI = 9);
      FN_R   : idSPDU := (SI = 9) AND     reuseTC(tsdu);
      FN_NR  : idSPDU := (SI = 9) AND NOT reuseTC(tsdu);

      DN     : idSPDU := (SI = 10);

      RF     : idSPDU := (SI = 12);
      RF_R   : idSPDU := (SI = 12) AND     reuseTC(tsdu);
      RF_NR  : idSPDU := (SI = 12) AND NOT reuseTC(tsdu);

      CN     : idSPDU := (SI = 13);
      AC     : idSPDU := (SI = 14);
      GTC    : idSPDU := (SI = 21);
      GTA    : idSPDU := (SI = 22);

      AB     : idSPDU := (SI = 25) AND Cat01;
      AB_R   : idSPDU := (SI = 25) AND Cat01 AND     reuseTC(tsdu);
      AB_NR  : idSPDU := (SI = 25) AND Cat01 AND NOT reuseTC(tsdu);

      AA     : idSPDU := (SI = 26) AND Cat01;
      AI     : idSPDU := (SI = 25) AND Cat2;
      AIA    : idSPDU := (SI = 26) AND Cat2;
      AR     : idSPDU := (SI = 29);
      AE     : idSPDU := (SI = 41);
      AEA    : idSPDU := (SI = 42);
      AS     : idSPDU := (SI = 45);
      ED     : idSPDU := (SI = 48);
      MIP    : idSPDU := (SI = 49);
      MIA    : idSPDU := (SI = 50);
      AD     : idSPDU := (SI = 57);
      ADA    : idSPDU := (SI = 58);
      END;

    END;
  ELSE                    {tsdu is empty}
    idSPDU := FALSE;
END;
```

```
{
 FUNCTION: Predicates

           These functions implement the Predicates specified in
           Rec. X.225 TABLE A-6/X.225.

           Only those Predicates required by X.400 are
           implemented.

 INPUTS:   (different Predicates have different inputs)
           n    - the Predicate selector.
           spsn - the Serial Number parameter value.
           rt   - the set of tokens requested in the input event.

 OUTPUTS:  returns: TRUE:  if the predicate is true,
                    FALSE: if false.

 CALLS:    I, II, A, AA, FU, AV, ALLT, ANYT.
}


PURE FUNCTION p(n : INTEGER) : BOOLEAN;

BEGIN
  CASE n OF
   1: p := NOT Vtca;
   2: p := REUSE_TC AND NOT Texp;
   3: p := I(DKT);
   5: p := A(DKT);
  11: p := II(MAT);
  14: p := (NOT FU(ACT) OR Vact) AND A(DKT) AND AA(MIT);
  15: p := (NOT FU(ACT) OR Vact) AND I(DKT) AND II(MIT);
  16: p := NOT Texp;
  17: p := (NOT FU(ACT) OR Vact) AND FU(SY) AND NOT Vsc;
  18: p := (NOT FU(ACT) OR Vact) AND FU(SY) AND Vsc;
  34: p := FU(ACT);
  38: p := FU(ACT) AND NOT Texp;
  39: p := Vact AND II(MAT);
  40: p := AA(MAT);
  44: p := FU(ACT) AND NOT Vact AND A(DKT) AND A(MIT) AND A(MAT);
  45: p := FU(ACT) AND NOT Vact AND I(DKT) AND I(MIT) AND I(MAT);
  48: p := FU(EXCEP) AND FU(HD);
  50: p := FU(EXCEP) AND (NOT FU(ACT) OR Vact) AND AA(DKT);
  51: p := FU(EXCEP) AND (NOT FU(ACT) OR Vact) AND II(DKT);
  55: p := FU(ACT) AND NOT Vact AND ALLT(I,TK_DOM);
  62: p := FU(ACT) AND NOT Vact AND ALLT(A,TK_DOM);
  63: p := ALLT(I,TK_DOM) AND (NOT FU(ACT) OR NOT Vact);
  64: p := REUSE_TC AND NOT Vtca AND NOT Texp;
  66: p := Vtrr;
  68: p := ALLT(A,TK_DOM) AND (NOT FU(ACT) OR NOT Vact);
  71: p := FU(ACT) AND Vact AND I(DKT) AND I(MIT) AND II(MAT);
  72: p := FU(ACT) AND Vact AND A(DKT) AND A(MIT) AND AA(MAT);
  END;
END;
```

```
PURE FUNCTION p19(spsn : INTEGER) : BOOLEAN;

BEGIN
  p19 := (spsn = Vm);
END;


PURE FUNCTION p20(spsn : INTEGER) : BOOLEAN;

BEGIN
  p20 := (spsn = (Vm-1));
END;


PURE FUNCTION p21(spsn : INTEGER) : BOOLEAN;

BEGIN
  p21 := (Vm > spsn) AND (spsn >= Va);
END;


PURE FUNCTION p53(rt : TokenSetTYPE) : BOOLEAN;

BEGIN
  p53 := ANYT(AV,rt);
END;
```

```
{
 PROCEDURE: Specific Actions

            These procedures implement the specific actions
            specified in Rec. X.225 TABLE A-5/X.225.

            Only those Specific Actions required by X.400 are
            implemented. Variables not required by X.400 are
            ignored.

 INPUTS:    (different Specific Actions have different inputs)
            n       - the Specific Action selector.
            tex     - selected Transport Expedited Data option.
            fus     - selected functional units.
            TCkept  - TC reuse option in SPDU.
            tokens  - set of owned tokens.
            spsn    - serial number.

 OUTPUTS:   external variables affected:
            Texp, Vact, Vnextact, Vtca, Vtrr, Va, Vm, Vsc,
            OwnedTokens, SelectedFUs.

            OUTPUT through TIMSAP: START, STOP.

 CALLS:     FU.
}


PROCEDURE SpAc(n : INTEGER);

BEGIN
  CASE n OF
   1 : Vtca := TRUE;
   2 : Vtca := FALSE;
   3 : OUTPUT TIMSAP.STOP;
   4 : OUTPUT TIMSAP.START(PERIOD);
   7 : Vtrr := TRUE;
   8 : Vtrr := FALSE;
  12 : Vact := TRUE;
  13 : IF FU(ACT) THEN Vnextact := FALSE;
  14 : Vact := Vnextact;
  22 : Va := Vm;
  23 : BEGIN
         IF NOT Vsc THEN Va := Vm;
         Vsc := TRUE;
         Vm := Vm + 1;
       END;
  24 : BEGIN
         IF Vsc THEN Va := Vm;
         Vsc:= FALSE;
         Vm := Vm + 1;
       END;
```

```
  26 : BEGIN
         Va := 1;
         Vm := 1;
       END;
  29 : BEGIN
         OwnedTokens := AvailableTokens;
         Vact := FALSE;
       END;
  30 : BEGIN
         OwnedTokens := [];
         Vact := FALSE;
       END;
  31 : BEGIN
         IF NOT Vsc THEN Va := Vm;
         Vm := Vm + 1;
       END;
  END;
END;
```

```
PROCEDURE SpAc5(spsn : INTEGER;
                tex  : BOOLEAN;
                fus  : FUsetTYPE);

BEGIN
  Va   := spsn;
  Vm   := spsn;
  Vsc  := FALSE;
  Texp := tex;
  SelectedFUs := fus;
  IF FU(ACT) THEN Vact := FALSE;
END;


PROCEDURE SpAc9(TCkept : BOOLEAN);

BEGIN
  Vtrr := REUSE_TC AND TCkept;
END;


PROCEDURE SpAc11(tokens : TokenSetTYPE);

BEGIN
  OwnedTokens := tokens;
END;


PROCEDURE SpAc25(spsn : INTEGER);

BEGIN
  Va := spsn + 1;
END;


PROCEDURE SpAc27(spsn : INTEGER);

BEGIN
  Va := spsn + 1;
  Vm := spsn + 1;
END;
```

```
{
 PROCEDURE: ProtocolErrorAbort

            This procedure is called upon detection of certain
            unrecoverable protocol errors.

 INPUTS:    none.

 OUTPUTS:   OUTPUT through SCEP: SPABind.
            OUTPUT through TCEP: TDTreq.

 CALLS:     BuildAB, SpAc.
}


PROCEDURE ProtocolErrorAbort;

VAR tsdu               : TSDUTYPE;
    ReflectParameters  : Bytes9TYPE;
    SSuserData         : Bytes9TYPE;
    TCdis              : TCdisTYPE;

BEGIN
  ReflectParameters.l := 0;
  SSuserData.l        := 0;

  TCdis.TCkept   := FALSE;          {for ABnr}
  TCdis.ABreason := PROTOCOL_ERROR; {reason for abort}

  BuildAB(tsdu,
          TCdis,
          ReflectParameters,
          SSuserData);

  OUTPUT TCEP.TDTreq(tsdu);

  OUTPUT SCEP.SPABind(PROTOCOL_ERROR);

  SpAc(4); {start timer TIM}
END;
```

```
{
 PROCEDURE: Abort

            This procedure is called upon legal reception of a
            AB SPDU.

 INPUTS:    TCkept - indicates whether or not the TC is to be
                     reused.

 OUTPUTS:   OUTPUT through SCEP: SPABind.
            OUTPUT through TCEP: TDTreq, TDISreq.

 CALLS:     StripAB, BuildAA.
}


PROCEDURE Abort(TCkept : BOOLEAN);

VAR TCdis              : TCdisTYPE;
    ReflectParameters  : Bytes9TYPE;
    SSuserData         : Bytes9TYPE;
    tsduAA             : TSDUTYPE;
    TSuserData         : Bytes64TYPE;

BEGIN
  TSuserData.l := 0;

  StripAB(tsdu,
          TCdis,
          ReflectParameters,
          SSuserData);

  {Generate event SUABind if TCdis.ABreason has the value
   "USER_ABORT". Otherwise, generate the event SPABind.}

  IF TCdis.ABreason = USER_ABORT
  THEN
    OUTPUT SCEP.SUABind(SSuserData);
  ELSE
    BEGIN
      CASE TCdis.ABreason OF
        PROTOCOL_ERROR : OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
        NO_REASON      : OUTPUT SCEP.SPABind(UNDEFINED);
      END;
    END;

  IF TCkept
  THEN
    BEGIN
      BuidAA(tsduAA);
      OUTPUT TCEP.TDTreq(tsduAA);
    END;
  ELSE
    OUTPUT TCEP.TDISreq(TSuserData);
END;
```

```
{
 PROCEDURE: UserAbort

            This procedure is called upon legal reception of a
            SUABreq primitive.

 INPUTS:    TCkept      - indicates whether or not the TC is to be
                          reused.
            SSuserData  - the SSuserData parameter value of the
                          SUABreq primitive.

 OUTPUTS:   OUTPUT through TCEP: TDTreq.

 CALLS:     BuildAB, SpAc.
}


PROCEDURE UserAbort(TCkept      : BOOLEAN;
                    SSuserData  : Bytes9TYPE);

VAR tsdu              : TSDUTYPE;
    TCdis             : TCdisTYPE;
    ReflectParameters : Bytes9TYPE;

BEGIN
  ReflectParameters.l := 0;

  TCdis.TCkept   := TCkept;     {for either ABr or ABnr}
  TCdis.ABreason := USER_ABORT; {reason for abort}

  BuildAB(tsdu,
          TCdis,
          ReflectParameters,
          SSuserData);

  OUTPUT TCEP.TDTreq(tsdu);

  SpAc(4); {start timer TIM}
END;
```

```
{*******************************
 initialization-part for SPMbody
 *******************************}

INITIALIZE

  TO STA01 {idle, no TC}

    BEGIN

      {
       Create one instance of the timer module and connect its
       external interaction point to SPMbody's internal
       interaction point.
      }

      INIT TimerInstance WITH TimerBody;

      CONNECT TIMSAP TO TimerInstance.TIMSAP;


      {initialization of 'constant' variables}


      TK_DOM := [DKT,MIT,MAT,TRT];

      FU_DOM := [HD,FD,EX,SY,MA,RESYN,ACT,NR,CD,EXCEP,TD];

      FU_SUP := [HD,SY,ACT,EXCEP];


      ALL token : TokenTYPE DO
        DEFAULT_TKNS[token] := REQUESTOR_SIDE;


      WITH DEFAULT_QOSS DO

        Protection                 := LEVEL_A;
        Priority                   := ANY PriorityTYPE;
        ResidualErrorRate          := ANY REAL;

        Throughput0                := ANY REAL;
        Throughput1                := ANY REAL;

        TransitDelay0              := ANY INTEGER;
        TransitDelay1              := ANY INTEGER;

        ExtendedControl            := FALSE;
        OptimizedDialogueTransfer  := FALSE;

      END;
```

```
  WITH DEFAULT_QOTS DO

    EstablishmentDelay                := ANY INTEGER;
    EstablishmentFailureProbability   := ANY REAL;

    Throughput0maximum                := ANY REAL;
    Throughput0average                := ANY REAL;
    Throughput1maximum                := ANY REAL;
    Throughput1average                := ANY REAL;

    TransitDelay0maximum              := ANY INTEGER;
    TransitDelay0average              := ANY INTEGER;
    TransitDelay1maximum              := ANY INTEGER;
    TransitDelay1average              := ANY INTEGER;

    ResidualErrorRate                 := ANY REAL;
    TransferFailureProbability        := ANY REAL;
    ReleaseDelay                      := ANY INTEGER;
    ReleaseFailureProbability         := ANY REAL;
    Protection                        := LEVEL_A;
    Priority                          := ANY PriorityTYPE;
    Resilience                        := ANY REAL;

  END;

END;
```

```
{****************************************
 transition-declaration-part for SPMbody
 ****************************************}


{
 Part 1:

 Transitions for handling valid intersections between SPM states
 and the following incoming events:

      i. SPDU events
     ii. SS-user events
    iii. TS-provider events
     iv. timer events


 Invalid SPDU event intersections are also handled here.
}
```

```
TRANS

FROM STA01 {transitions from STA01 - idle, no TC}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    TO SAME

    VAR TSuserData : Bytes64TYPE;

    BEGIN
      TSuserData.l := 0;
      OUTPUT TCEP.TDISreq(TSuserData);
    END;
```

```
{SS-user events}

WHEN SCEP.SCONreq(CallingSSuserRef,
                  CommonRef,
                  AdditionalRef,
                  CallingSSAPaddr,
                  CalledSSAPaddr,
                  qossReq,
                  Srequirements,
                  InitialSpsn,
                  InitialTokens,
                  SSuserData)

  TO STA01B {await TCONcnf}

  VAR CallingSSAPid : Bytes16TYPE;
      CalledSSAPid  : Bytes16TYPE;
      TSuserData    : Bytes32TYPE;
      ProposedTEXP  : BOOLEAN;

  BEGIN
    CallingSSAPid.l := 0;
    CalledSSAPid.l  := 0;
    TSuserData.l    := 0;

    LocalAddress  := CallingSSAPaddr;
    RemoteAddress := CalledSSAPaddr;
    SelectedFUs   := Srequirements;

    BuildCN(TempTSDU,
            CallingSSuserRef,
            CommonRef,
            AdditionalRef,
            PROTOCOL,
            MAXTSDULEN,
            MAXTSDULEN,
            VERSION,
            InitialSpsn,
            InitialTokens,
            Srequirements,
            CallingSSAPid,
            CalledSSAPid,
            SSuserData);

    {Determine whether the Transport Expedited Data option
     must be requested for this TC or not: ('FALSE' for X.400)}
    ProposedTEXP := qossReq.ExtendedControl OR
                    (EX IN Srequirements);

    OUTPUT TCEP.TCONreq(LocalAddress,
                        RemoteAddress,
                        ProposedTEXP,
                        DEFAULT_QOTS,
                        TSuserData);
    SpAc(2);
  END;
```

```
{TS-provider events}

WHEN TCEP.TCONind(CallingTSAPaddr,
                  CalledTSAPaddr,
                  ProposedTEXP,
                  qots,
                  TSuserData)

  TO STA01C {idle, TC con}

  VAR TSuserDataRsp : Bytes32TYPE;

  BEGIN
    TSuserDataRsp.1 := 0;

    RemoteAddress := CallingTSAPaddr;
    LocalAddress  := CalledTSAPaddr;
    SelectedQOTS  := qots;

    {Determine whether the Transport Expedited Data option
     will be available to this TC or not: ('FALSE' for X.400)}

    Texp := TEXP_LOCAL AND ProposedTEXP;

    OUTPUT TCEP.TCONrsp(LocalAddress;
                        Texp,
                        SelectedQOTS,
                        TSuserDataRsp);

    SpAc(1);
  END;
```

```
FROM STA01A {transitions from STA01A - await AA}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,DT)  OR
             idSPDU(tsdu,PT)  OR
             idSPDU(tsdu,FN)  OR
             idSPDU(tsdu,DN)  OR
             idSPDU(tsdu,RF)  OR
             idSPDU(tsdu,AC)  OR
             idSPDU(tsdu,GTC) OR
             idSPDU(tsdu,GTA) OR
             idSPDU(tsdu,AI)  OR
             idSPDU(tsdu,AIA) OR
             idSPDU(tsdu,AR)  OR
             idSPDU(tsdu,AE)  OR
             idSPDU(tsdu,AEA) OR
             idSPDU(tsdu,AS)  OR
             idSPDU(tsdu,ED)  OR
             idSPDU(tsdu,MIP) OR
             idSPDU(tsdu,MIA) OR
             idSPDU(tsdu,AD)  OR
             idSPDU(tsdu,ADA)

      TO SAME

      BEGIN
      END;


    PROVIDED idSPDU(tsdu,CN)    OR
             idSPDU(tsdu,AB_NR)

      TO STA01 {idle, no TC}

      VAR TSuserData : Bytes64TYPE;

      BEGIN
        TSuserData.l := 0;
        OUTPUT TCEP.TDISreq(TSuserData);
        SpAc(3);
      END;


    PROVIDED idSPDU(tsdu,AA)   OR
             idSPDU(tsdu,AB_R)

      TO STA01C {idle, TC con}

      BEGIN
        SpAc(3);
      END;
```

```
  PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

  TO STA16 {await TDISind}

  BEGIN
    ProtocolErrorAbort;
  END;


{TSprovider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
    SpAc(3);
  END;


{timer events}

WHEN TIMSAP.TIMEOUT

  TO STA01 {idle, no TC}

  VAR TSuserData : Bytes64TYPE;

  BEGIN
    TSuserData.l := 0;
    OUTPUT TCEP.TDISreq(TSuserData);
  END;
```

```
FROM STA01B {transitions from STA01B - await TCONcnf}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    TO STA01 {idle, no TC}

    VAR TSuserData : Bytes64TYPE;

    BEGIN
      TSuserData.l := 0;
      OUTPUT TCEP.TDISreq(TSuserData);
    END;

  {SS-user events}

  WHEN SCEP.SUABreq(SSuserData)

    TO STA01 {idle, no TC}

    VAR TSuserData : Bytes64TYPE;

    BEGIN
      TSuserData.l := 0;
      OUTPUT TCEP.TDISreq(TSuserData);
    END;
```

```
{TS-provider events}

WHEN TCEP.TCONcnf(RespondTSAPaddr,
                  SelectedTEXP,
                  qots,
                  TSuserData)

  TO STA02A {await AC}

  BEGIN
    Texp         := SelectedTEXP;
    SelectedQOTS := qots;

    OUTPUT TCEP.TDTreq(TempTSDU); {CN already built in TempTSDU}
  END;


WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
    OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
  END;
```

```
FROM STA01C {transitions from STA01C - idle, TC con}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,DT)    OR
             idSPDU(tsdu,PT)    OR
             idSPDU(tsdu,FN)    OR
             idSPDU(tsdu,DN)    OR
             idSPDU(tsdu,RF)    OR
             idSPDU(tsdu,CN) AND p(1) OR
             idSPDU(tsdu,AC)    OR
             idSPDU(tsdu,GTC)   OR
             idSPDU(tsdu,GTA)   OR
             idSPDU(tsdu,AI)    OR
             idSPDU(tsdu,AA)    OR
             idSPDU(tsdu,AIA)   OR
             idSPDU(tsdu,AR)    OR
             idSPDU(tsdu,AE)    OR
             idSPDU(tsdu,AEA)   OR
             idSPDU(tsdu,AS)    OR
             idSPDU(tsdu,ED)    OR
             idSPDU(tsdu,MIP)   OR
             idSPDU(tsdu,MIA)   OR
             idSPDU(tsdu,AD)    OR
             idSPDU(tsdu,ADA)   OR
             idSPDU(tsdu,AB_NR) OR
             idSPDU(tsdu,AB_R) AND NOT p(2)

      TO STA01 {idle, no TC}

      VAR TSuserData : Bytes64TYPE;

      BEGIN
        TSuserData.l := 0;
        OUTPUT TCEP.TDISreq(TSuserData);
      END;
```

```
PROVIDED idSPDU(tsdu,CN) AND NOT p(1)

  TO STA08  {await SCONrsp}

  VAR CallingSSuserRef : Bytes64TYPE;
      CommonRef        : Bytes64TYPE;
      AdditionalRef    : Bytes4TYPE;
      ProtocolOptions  : ByteTYPE;
      maxTSDUlen0      : INTEGER;
      maxTSDUlen1      : INTEGER;
      VersionNumber    : ByteTYPE;
      InitialSpsn      : INTEGER;
      TokenSettingItem : InitialTokensTYPE;
      Srequirements    : FUsetTYPE;
      CallingSSAPid    : Bytes16TYPE;
      CalledSSAPid     : Bytes16TYPE;
      SSuserData       : Bytes512TYPE;

  BEGIN
    StripCN(tsdu,
            CallingSSuserRef,
            CommonRef,
            AdditionalRef,
            ProtocolOptions,
            maxTSDUlen0,
            maxTSDUlen1,
            VersionNumber,
            InitialSpsn,
            TokenSettingItem,
            Srequirements,
            CallingSSAPid,
            CalledSSAPid,
            SSuserData);

    Protocol    := ProtocolOptions;
    TempTokens  := TokenSettingItem; {used when SCONrsp+}
    SelectedFUs := Srequirements;

    {negotiate max TSDU length for each transfer direction}

    IF maxTSDUlen0 < MAXTSDULEN }initiator to responder}
    THEN
      MaxTSDU0 := maxTSDUlen0
    ELSE
      MaxTSDU0 := MAXTSDULEN;

    IF maxTSDUlen1 < MAXTSDULEN {responder to initiator}
    THEN
      MaxTSDU1 := maxTSDUlen1
    ELSE
      MaxTSDU1 := MAXTSDULEN;
```

```
    IF VersionNumber < VERSION }negotiate version number}
    THEN
      Version := VersionNumber
    ELSE
      Version := VERSION;

    OUTPUT SCEP.SCONind(CallingSSuserRef,
                        CommonRef,
                        AdditionalRef,
                        RemoteAddress,
                        LocalAddress,
                        DEFAULT_QOSS,
                        Srequirements,
                        InitialSpsn,
                        TokenSettingItem,
                        SSuserData);
  END;


PROVIDED idSPDU(tsdu,AB_R) AND p(2)

  TO SAME

  VAR tsdu : TSDUTYPE;

  BEGIN
    BuildAA(tsdu);
    OUTPUT TCEP.TDTreq(tsdu);
  END;


PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

  TO STA01 {idle, no TC}

  VAR TSuserData : Bytes64TYPE;

  BEGIN
    TSuserData.l := 0;
    OUTPUT TCEP.TDISreq(TSuserData);
  END;
```

```
{SS-user events}

WHEN SCEP.SCONreq(CallingSSuserRef,
                  CommonRef,
                  AdditionalRef,
                  CallingSSAPaddr,
                  CalledSSAPaddr,
                  qossReq,
                  Srequirements,
                  InitialSpsn,
                  InitialTokens,
                  SSuserData)

  PROVIDED p(1)

    TO STA02A {await AC}

    VAR tsdu           : TSDUTYPE;
        CallingSSAPid  : Bytes16TYPE;
        CalledSSAPid   : Bytes16TYPE;

    BEGIN
      CallingSSAPid.1 := 0;
      CalledSSAPid.1  := 0;

      LocalAddress  := CallingSSAPaddr;
      RemoteAddress := CalledSSAPaddr;
      SelectedFUs   := Srequirements;

      BuildCN(tsdu,
              CallingSSuserRef,
              CommonRef,
              AdditionalRef,
              PROTOCOL,
              MAXTSDULEN,
              MAXTSDULEN,
              VERSION,
              InitialSpsn,
              InitialTokens,
              Srequirements,
              CallingSSAPid,
              CalledSSAPid,
              SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);
    END;


  PROVIDED OTHERWISE

    TO SAME

    BEGIN
      SPABind(PROTOCOL_ERROR);
    END;
```

```
{TS-provider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
  END;
```

```
FROM STA02A {transitions from STA02A - await AC}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,RF_NR) OR
             idSPDU(tsdu,RF_R)  AND NOT p(2)

      TO STA01 {idle, no TC}

      VAR CalledSSuserRef : Bytes64TYPE;
          CommonRef       : Bytes64TYPE;
          AdditionalRef   : Bytes4TYPE;
          TCdis           : TCdisTYPE;
          Srequirements   : FUsetTYPE;
          VersionNumber   : ByteTYPE;
          ReasonCode      : ReasonCodeTYPE;

          TSuserData      : Bytes64TYPE;

      BEGIN
        TSuserData.l := 0;

        StripRF(tsdu,
                CalledSSuserRef,
                CommonRef,
                AdditionalRef,
                TCdis,
                Srequirements,
                VersionNumber,
                ReasonCode);

        OUTPUT SCEP.SCONcnf(CalledSSuserRef,
                            CommonRef,
                            AdditionalRef,
                            RemoteAddress,
                            MapRefCnf(ReasonCode.Reason),
                            DEFAULT_QOSS,
                            Srequirements,
                            DEFAULT_SPSN,
                            DEFAULT_TKNS,
                            ReasonCode.Data); {SS-user data}

        OUTPUT TCEP.TDISreq(TSuserData);
      END;
```

```
PROVIDED idSPDU(tsdu,RF_R) AND p(2)

  TO STA01C {idle, TC con}

  VAR CalledSSuserRef : Bytes64TYPE;
      CommonRef       : Bytes64TYPE;
      AdditionalRef   : Bytes4TYPE;
      TCdis           : TCdisTYPE;
      Srequirements   : FUsetTYPE;
      VersionNumber   : ByteTYPE;
      ReasonCode      : ReasonCodeTYPE;

  BEGIN
    StripRF(tsdu,
            CalledSSuserRef,
            CommonRef,
            AdditionalRef,
            TCdis,
            Srequirements,
            VersionNumber,
            ReasonCode);

    OUTPUT SCEP.SCONcnf(CalledSSuserRef,
                        CommonRef,
                        AdditionalRef,
                        RemoteAddress,
                        MapRefCnf(ReasonCode.Reason),
                        DEFAULT_QOSS,
                        Srequirements,
                        DEFAULT_SPSN,
                        DEFAULT_TKNS,
                        ReasonCode.Data); {SS-user data}
  END;
```

```
PROVIDED idSPDU(tsdu,AC)

  TO STA713 {data transfer}

  VAR CalledSSuserRef  : Bytes64TYPE;
      CommonRef        : Bytes64TYPE;
      AdditionalRef    : Bytes4TYPE;
      ProtocolOptions  : ByteTYPE;
      maxTSDUlen0      : INTEGER;
      maxTSDUlen1      : INTEGER;
      VersionNumber    : ByteTYPE;
      InitialSpsn      : INTEGER;
      TokenSettingItem : InitialTokensTYPE;
      TokenItem        : TokenSetTYPE;
      Srequirements    : FUsetTYPE;
      CallingSSAPid    : Bytes16TYPE;
      CalledSSAPid     : Bytes16TYPE;
      SSuserData       : Bytes512TYPE;

      SPTdata          : Bytes512TYPE;

  BEGIN
    SPTdata.l := 0;

    StripAC(tsdu,
            CalledSSuserRef,
            CommonRef,
            AdditionalRef,
            ProtocolOptions,
            maxTSDUlen0,
            maxTSDUlen1,
            VersionNumber,
            InitialSpsn,
            TokenSettingItem,
            TokenItem,
            Srequirements,
            CallingSSAPid,
            CalledSSAPid,
            SSuserData);

    SelectedQOSS := DEFAULT_QOSS;
    Protocol     := ProtocolOptions;

    {negotiate max TSDU length for each transfer direction}

    IF maxTSDUlen0 < MAXTSDULEN {initiator to responder}
    THEN
      MaxTSDU0 := maxTSDUlen0;
    ELSE
      MaxTSDU0 := MAXTSDULEN;

    IF maxTSDUlen1 < MAXTSDULEN {responder to initiator}
    THEN
      MaxTSDU1 := maxTSDUlen1;
    ELSE
      MaxTSDU1 := MAXTSDULEN;
```

```
    IF VersionNumber < VERSION {negotiate version number}
    THEN
      Version := VersionNumber
    ELSE
      Version := VERSION;

    {The functional units selected for this SC}
    SelectedFUs := SelectedFUs * Srequirements;

    {The set of tokens available on this SC}
    AvailableTokens := [];
    ALL token : TokenTYPE DO
      IF AV(token)
      THEN
        AvailableTokens := AvailableTokens + [token];

    {The set of owned tokens}
    OwnedTokens := [];
    ALL token : TokenTYPE DO
      IF AV(token) AND
         (TokenSettingItem[token] = REQUESTOR_SIDE)
      THEN
        OwnedTokens := OwnedTokens + [token];

    OUTPUT SCEP.SCONcnf(CalledSSuserRef,
                        CommonRef,
                        AdditionalRef,
                        RemoteAddress,
                        ACCEPT,            {for SCONcnf+}
                        DEFAULT_QOSS,
                        Srequirements,
                        InitialSpsn,
                        TokenSettingItem,
                        SSuserData);

    {Issue SPTind if the called SS-user requests any tokens}
    IF TokenItem <> []
    THEN
      OUTPUT SCEP.SPTind(TokenItem,
                         SPTdata);

    SpAc5(InitialSpsn,Texp,SelectedFUs);
    SpAc11(OwnedTokens);
  END;
```

```
PROVIDED idSPDU(tsdu,AB_NR) OR
         idSPDU(tsdu,AB_R)  AND NOT p(2)

  TO STA01 {idle, no TC}

  BEGIN
    Abort(FALSE);
  END;


PROVIDED idSPDU(tsdu,AB_R) AND p(2)

  TO STA01C {idle, TC con}

  BEGIN
    Abort(TRUE);
  END;


PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

  TO STA16 {await TDISind}

  BEGIN
    ProtocolErrorAbort;
  END;
```

```
{SS-user events}

WHEN SCEP.SUABreq(SSuserData)

  PROVIDED NOT p(2)

    TO STA16 {await TDISind}

    BEGIN
      UserAbort(FALSE,SSuserData);
    END;


  PROVIDED p(2)

    TO STA01A {await AA}

    BEGIN
      UserAbort(TRUE,SSuserData);
    END;


{TS-provider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
    OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
  END;
```

```
FROM STA03 {transitions from STA03 - await DN}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,PT) AND p53(TokensVal(tsdu))

      TO SAME

      VAR TokenItem  : TokenSetTYPE;
          SSuserData : Bytes512TYPE;

      BEGIN
        StripPT(tsdu,
                TokenItem,
                SSuserData);

        OUTPUT SCEP.SPTind(TokenItem,
                           SSuserData);
      END;

    PROVIDED idSPDU(tsdu,DN) AND NOT p(66)

      TO STA01 {idle, no TC}

      VAR SSuserData : Bytes512TYPE;
          TSuserData : Bytes64TYPE;

      BEGIN
        TSuserData.l := 0;

        StripDN(tsdu,
                SSuserData);

        OUTPUT SCEP.SRELcnf(AFFIRMATIVE, {for SRELcnf+},
                            SSuserData);

        OUTPUT TCEP.TDISreq(TSuserData);
      END;
```

```
PROVIDED idSPDU(tsdu,DN) AND p(66)

  TO STA01C {idle, TC con}

  VAR SSuserData : Bytes512TYPE;

  BEGIN
    StripDN(tsdu,
            SSuserData);

    OUTPUT SCEP.SRELcnf(AFFIRMATIVE, {for SRELcnf+},
                        SSuserData);
  END;


PROVIDED idSPDU(tsdu,AB_NR) OR
         idSPDU(tsdu,AB_R)  AND NOT p(2)

  TO STA01 {idle, no TC}

  BEGIN
    Abort(FALSE);
  END;


PROVIDED idSPDU(tsdu,AB_R) AND p(2)

  TO STA01C {idle, TC con}

  BEGIN
    Abort(TRUE);
  END;


PROVIDED idSPDU(tsdu,ED) AND p(52)

  TO STA20 {await recovery SPDU or request}

  VAR ReasonCode : ReasonCodeTYPE;
      SSuserData : Bytes512TYPE;

  BEGIN
    StripED(tsdu,
            ReasonCode,
            SSuserData);

    OUTPUT SCEP.SUERind(MapErActInd(ReasonCode.Reason),
                        SSuserData);
  END;
```

```
PROVIDED idSPDU(tsdu,MIA) AND p(17) AND p21(SpsnVal(tsdu))

  TO SAME

  VAR spsn       : INTEGER;
      SSuserData : Bytes512TYPE;

  BEGIN
    StripMIA(tsdu,
             spsn,
             SSuserData);

    OUTPUT SCEP.SSYNmcnf(spsn,
                         SSuserData);

    SpAc25(spsn);
  END;


PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

  TO STA16 {await TDISind}

  BEGIN
    ProtocolErrorAbort;
  END;
```

```
{SS-user events}

WHEN SCEP.SUABreq(SSuserData)

  PROVIDED NOT p(2)

    TO STA16 {await TDISind}

    BEGIN
      UserAbort(FALSE,SSuserData);
    END;


  PROVIDED p(2)

    TO STA01A {await AA}

    BEGIN
      UserAbort(TRUE,SSuserData);
    END;

{TS-provider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
    OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
  END;
```

```
FROM STA04B {transitions from STA04B - await AEA}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,PT) AND p53(TokensVal(tsdu))

      TO SAME

      VAR TokenItem  : TokenSetTYPE;
          SSuserData : Bytes512TYPE;

      BEGIN
        StripPT(tsdu,
                TokenItem,
                SSuserData);

        OUTPUT SCEP.SPTind(TokenItem,
                           SSuserData);
      END;


    PROVIDED idSPDU(tsdu,AB_NR) OR
             idSPDU(tsdu,AB_R)  AND NOT p(2)

      TO STA01 {idle, no TC}

      BEGIN
        Abort(FALSE);
      END;


    PROVIDED idSPDU(tsdu,AB_R) AND p(2)

      TO STA01C {idle, TC con}

      BEGIN
        Abort(TRUE);
      END;
```

```
PROVIDED idSPDU(tsdu,AEA) AND p(16) AND p20(SpsnVal(tsdu))

  TO STA713 {data transfer}

  VAR spsn       : INTEGER;
      SSuserData : Bytes512TYPE;

  BEGIN
    StripAEA(tsdu,
             spsn,
             SSuserData);

    OUTPUT SCEP.SACTEcnf(SSuserData);

    SpAc(14);
    SpAc(22);
  END;


PROVIDED idSPDU(tsdu,ED) AND p(48)

  TO STA20 {await recovery SPDU or request}

  VAR ReasonCode : ReasonCodeTYPE;
      SSuserData : Bytes512TYPE;

  BEGIN
    StripED(tsdu,
            ReasonCode,
            SSuserData);

    OUTPUT SCEP.SUERind(MapErActInd(ReasonCode.Reason),
                        SSuserData);
  END;


PROVIDED idSPDU(tsdu,MIA) AND p(17) AND NOT p20(SpsnVal(tsdu))
         AND p21(SpsnVal(tsdu))

  TO SAME

  VAR spsn       : INTEGER;
      SSuserData : Bytes512TYPE;

  BEGIN
    StripMIA(tsdu,
             spsn,
             SSuserData);

    OUTPUT SCEP.SSYNmcnf(spsn,
                         SSuserData);

    SpAc25(spsn);
  END;
```

```
  PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;


{SS-user events}

WHEN SCEP.SACTIreq(Reason)

  PROVIDED p(39)

    TO STA05B {await AIA}

    VAR ReasonCode : ReasonCodeTYPE;
        tsdu       : TSDUTYPE;

    BEGIN
      ReasonCode.Data.l := 0;
      ReasonCode.Reason := MapErActReq(Reason);

      BuildAI(tsdu,
              ReasonCode);

      OUTPUT TCEP.TDTreq(tsdu);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;
```

```
WHEN SCEP.SACTDreq(Reason)

  PROVIDED p(39)

    TO STA05C {await ADA}

    VAR ReasonCode : ReasonCodeTYPE;
        tsdu       : TSDUTYPE;

    BEGIN
      ReasonCode.Data.1 := 0;
      ReasonCode.Reason := MapErActReq(Reason);

      BuildAD(tsdu,
              ReasonCode);

      OUTPUT TCEP.TDTreq(tsdu);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;


WHEN SCEP.SUABreq(SSuserData)

  PROVIDED NOT p(2)

    TO STA16 {await TDISind}

    BEGIN
      UserAbort(FALSE,SSuserData);
    END;


  PROVIDED p(2)

    TO STA01A {await AA}

    BEGIN
      UserAbort(TRUE,SSuserData);
    END;
```

```
{TS-provider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
    OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
  END;
```

```
FROM STA05B {transitions from STA05B - await AIA}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,PT)  AND p53(TokensVal(tsdu)) OR
             idSPDU(tsdu,AEA)                          OR
             idSPDU(tsdu,ED)  AND p(48)                OR
             idSPDU(tsdu,MIA) AND p(17)

      TO SAME

      BEGIN
      END;


    PROVIDED idSPDU(tsdu,AB_NR) OR
             idSPDU(tsdu,AB_R)  AND NOT p(2)

      TO STA01 {idle, no TC}

      BEGIN
        Abort(FALSE);
      END;


    PROVIDED idSPDU(tsdu,AB_R) AND p(2)

      TO STA01C {idle, TC con}

      BEGIN
        Abort(TRUE);
      END;


    PROVIDED idSPDU(tsdu,AIA) AND p(38)

      TO STA713 {data transfer}

      BEGIN
        OUTPUT SCEP.SACTIcnf;
        SpAc(29);
      END;


    PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

      TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
{SS-user events}

WHEN SCEP.SUABreq(SSuserData)

  PROVIDED NOT p(2)

    TO STA16 {await TDISind}

    BEGIN
      UserAbort(FALSE,SSuserData);
    END;

  PROVIDED p(2)

    TO STA01A {await AA}

    BEGIN
      UserAbort(TRUE,SSuserData);
    END;

{TS-provider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
    OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
  END;
```

```
FROM STA05C {transitions from STA05C - await ADA}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,PT)  AND p53(TokensVal(tsdu)) OR
             idSPDU(tsdu,AEA)                          OR
             idSPDU(tsdu,ED)  AND p(48)                OR
             idSPDU(tsdu,MIA) AND p(17)

      TO SAME

      BEGIN
      END;

    PROVIDED idSPDU(tsdu,AB_NR) OR
             idSPDU(tsdu,AB_R)  AND NOT p(2)

      TO STA01 {idle, no TC}

      BEGIN
        Abort(FALSE);
      END;

    PROVIDED idSPDU(tsdu,AB_R) AND p(2)

      TO STA01C {idle, TC con}

      BEGIN
        Abort(TRUE);
      END;

    PROVIDED idSPDU(tsdu,ADA) AND p(38)

      TO STA713 {data transfer}

      BEGIN
        OUTPUT SCEP.SACTDcnf;
        SpAc(29);
      END;

    PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

      TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
{SS-user events}

WHEN SCEP.SUABreq(SSuserData)

  PROVIDED NOT p(2)

    TO STA16 {await TDISind}

    BEGIN
      UserAbort(FALSE,SSuserData);
    END;


  PROVIDED p(2)

    TO STA01A {await AA}

    BEGIN
      UserAbort(TRUE,SSuserData);
    END;


{TS-provider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
    OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
  END;
```

```
FROM STA08 {transitions from STA08 - await SCONrsp}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,AB_NR) OR
             idSPDU(tsdu,AB_R)  AND NOT p(2)

      TO STA01 {idle, no TC}

      BEGIN
        Abort(FALSE);
      END;

    PROVIDED idSPDU(tsdu,AB_R) AND p(2)

      TO STA01C {idle, TC con}

      BEGIN
        Abort(TRUE);
      END;

    PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

      TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
{SS-user events}

WHEN SCEP.SCONrsp(CalledSSuserRef,
                  CommonRef,
                  AdditionalRef,
                  CalledSSAPaddr,
                  Result,
                  qoss,
                  Srequirements,
                  InitialSpsn,
                  InitialTokens,
                  SSuserData)

  PROVIDED Result = ACCEPT {SCONrsp+}

    TO STA713 {data transfer}

    VAR TokenSettingItem : InitialTokensTYPE;
        CallingSSAPid    : Bytes16TYPE;
        CalledSSAPid     : Bytes16TYPE;

        tsdu             : TSDUTYPE;

    BEGIN
      CallingSSAPid.1 := 0;
      CalledSSAPid.1  := 0;

      SelectedQOSS := DEFAULT_QOSS;

      {The functional units selected for this SC}
      SelectedFUs := SelectedFUs * Srequirements;

      {The set of available tokens}
      AvailableTokens := [];
      ALL token : TokenTYPE DO
        IF AV(token)
        THEN
          AvailableTokens := AvailableTokens + [token];

      {Determine TokenSettingItem for AC and
       the set OwnedTokens}
      OwnedTokens := [];
      ALL token : TokenTYPE DO
        BEGIN
          IF TempTokens[token] = ACCEPTOR_CHOOSES
          THEN
            TokenSettingItem[token] := InitialTokens[token];
          ELSE
            TokenSettingItem[token] := TempTokens[token];

          IF AV(token) AND
             (TokenSettingItem[token] = ACCEPTOR_SIDE)
          THEN
            OwnedTokens := OwnedTokens + [token];
        END;
```

```
  BuildAC(tsdu,
          CalledSSuserRef,
          CommonRef,
          AdditionalRef,
          PROTOCOL,
          MAXTSDUEN,
          MAXTSDUEN,
          VERSION,
          InitialSpsn,
          TokenSettingItem,
          [],                {null Token Item}
          Srequirements,
          CallingSSAPid,
          CalledSSAPid,
          SSuserData);

  OUTPUT TCEP.TDTreq(tsdu);

  SpAc5(InitialSpsn,Texp,SelectedFUs);
  SpAc11(OwnedTokens);
END;
```

```
PROVIDED (Result <> ACCEPT) AND NOT p(2)

  TO STA16 {await TDISind}

  VAR TCdis      : TCdisTYPE;
      ReasonCode : ReasonCodeTYPE;
      tsdu       : TSDUTYPE;

  BEGIN
    TCdis.TCkept   := FALSE;    {for RFnr}
    TCdis.ABreason := NO_ABORT; {No Abort in progress}

    ReasonCode.Data   := SSuserData;
    ReasonCode.Reason := MapRefRsp(Result);

    BuildRF(tsdu,
            CalledSSuserRef,
            CommonRef,
            AdditionalRef,
            TCdis,
            Srequirements,
            VERSION,
            ReasonCode);

    OUTPUT TCEP.TDTreq(tsdu);

    SpAc(4);
  END;

PROVIDED (Result <> ACCEPT) AND p(2)

  TO STA01C {idle, TC con}

  VAR TCdis      : TCdisTYPE;
      ReasonCode : ReasonCodeTYPE;
      tsdu       : TSDUTYPE;

  BEGIN
    TCdis.TCkept   := TRUE;     {for RFr}
    TCdis.ABreason := NO_ABORT; {No Abort in progress}

    ReasonCode.Data   := SSuserData;
    ReasonCode.Reason := MapRefRsp(Result);

    BuildRF(tsdu,
            CalledSSuserRef,
            CommonRef,
            AdditionalRef,
            TCdis,
            Srequirements,
            VERSION,
            ReasonCode);

    OUTPUT TCEP.TDTreq(tsdu);
  END;
```

```
WHEN SCEP.SUABreq(SSuserData)

  PROVIDED NOT p(2)

    TO STA16 {await TDISind}

    BEGIN
      UserAbort(FALSE,SSuserData);
    END;


  PROVIDED p(2)

    TO STA01A {await AA}

    BEGIN
      UserAbort(TRUE,SSuserData);
    END;


{TS-provider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
    OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
  END;
```

```
FROM STA09 {transitions from STA09 - await SRELrsp}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,AB_NR) OR
             idSPDU(tsdu,AB_R)  AND NOT p(2)

      TO STA01 {idle, no TC}

      BEGIN
        Abort(FALSE);
      END;

    PROVIDED idSPDU(tsdu,AB_R) AND p(2)

      TO STA01C {idle, TC con}

      BEGIN
        Abort(TRUE);
      END;

    PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

      TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
{SS-user events}

WHEN SCEP.SPTreq(Tokens,
                 SSuserData)

  PROVIDED p53(Tokens)

    TO SAME

    VAR tsdu : TSDUTYPE;

    BEGIN
      BuildPT(tsdu,
              Tokens,
              SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;


WHEN SCEP.SSYNmrsp(spsn,
                   SSuserData)

  PROVIDED p(18) AND p21(spsn)

    TO SAME

    VAR tsdu : TSDUTYPE;

    BEGIN
      BuildMIA(tsdu,
               spsn,
               SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);

      SpAc25(spsn);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;
```

```
WHEN SCEP.SUERreq(Reason,
                  SSuserData)

  PROVIDED p(50)

    TO STA19 {await recovery request or SPDU}

    VAR ReasonCode : ReasonCodeTYPE;
        tsdu       : TSDUTYPE;

    BEGIN
      ReasonCode.Data.l := 0;
      ReasonCode.Reason := MapErActReq(Reason);

      BuildED(tsdu,
              ReasonCode,
              SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;


WHEN SCEP.SRELrsp(Result,
                  SSuserData)

  PROVIDED (Result = AFFIRMATIVE) AND NOT p(66)

    TO STA16 {await TDISind}

    VAR tsdu : TSDUTYPE;

    BEGIN
      BuildDN(tsdu,
              SSuserData);

      OUTPUT TSAP.TDTreq(tsdu);

      SpAc(4);
    END;
```

```
  PROVIDED (Result = AFFIRMATIVE) AND p(66)

    TO STA01C {idle, TC con}

    VAR tsdu : TSDUTYPE;

    BEGIN
      BuildDN(tsdu,
              SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;


WHEN SCEP.SUABreq(SSuserData)

  PROVIDED NOT p(2)

    TO STA16 {await TDISind}

    BEGIN
      UserAbort(FALSE,SSuserData);
    END;


  PROVIDED p(2)

    TO STA01A {await AA}

    BEGIN
      UserAbort(TRUE,SSuserData);
    END;


{TS-provider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
    OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
  END;
```

```
FROM STA10B {transitions from STA10B - await SACTErsp}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,AB_NR) OR
             idSPDU(tsdu,AB_R)  AND NOT p(2)

      TO STA01 {idle, no TC}

      BEGIN
        Abort(FALSE);
      END;


    PROVIDED idSPDU(tsdu,AB_R) AND p(2)

      TO STA01C {idle, TC con}

      BEGIN
        Abort(TRUE);
      END;


    PROVIDED idSPDU(tsdu,AI) AND p(38) AND p(40)

      TO STA11B {await SACTIrsp}

      VAR ReasonCode : ReasonCodeTYPE;

      BEGIN
        StripAI(tsdu,
                ReasonCode);

        OUTPUT SCEP.SACTIind(MapErActInd(ReasonCode.Reason));
      END;


    PROVIDED idSPDU(tsdu,AD) AND p(38) AND p(40)

      TO STA11C {await SACTDrsp}

      VAR ReasonCode : ReasonCodeTYPE;

      BEGIN
        StripAD(tsdu,
                ReasonCode);

        OUTPUT SCEP.SACTDind(MapErActInd(ReasonCode.Reason));
      END;
```

```
PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

  TO STA16 {await TDISind}

  BEGIN
    ProtocolErrorAbort;
  END;
```

```
{SS-user events}

WHEN SCEP.SPTreq(Tokens,
                 SSuserData)

  PROVIDED p53(Tokens)

    TO SAME

    VAR tsdu : TSDUTYPE;

    BEGIN
      BuildPT(tsdu,
              Tokens,
              SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;


WHEN SCEP.SSYNmrsp(spsn,
                   SSuserData)

  PROVIDED p(18) AND NOT p20(spsn) AND p21(spsn)

    TO SAME

    VAR TSDU : TSDUTYPE;

    BEGIN
      BuildMIA(tsdu,
               spsn,
               SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);

      SpAc25(spsn);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;
```

```
WHEN SCEP.SUERreq(Reason,
                  SSuserData)

  PROVIDED p(50)

    TO STA19 {await recovery request or SPDU}

    VAR ReasonCode : ReasonCodeTYPE;
        tsdu       : TSDUTYPE;

    BEGIN
      ReasonCode.Data.l := 0;
      ReasonCode.Reason := MapErActReq(Reason);

      BuildED(tsdu,
              ReasonCode,
              SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;


WHEN SCEP.SACTErsp(SSuserData)

  TO STA713 {data transfer}

  VAR tsdu : TSDUTYPE;

  BEGIN
    BuildAEA(tsdu,
             Vm-1,        {serial number}
             SSuserData);

    OUTPUT TCEP.TDTreq(tsdu);

    SpAc(14);
    SpAc(22);
  END;
```

```
WHEN SCEP.SUABreq(SSuserData)

  PROVIDED NOT p(2)

    TO STA16 {await TDISind}

    BEGIN
      UserAbort(FALSE,SSuserData);
    END;


  PROVIDED p(2)

    TO STA01A {await AA}

    BEGIN
      UserAbort(TRUE,SSuserData);
    END;

{TS-provider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
    OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
  END;
```

```
FROM STA11B {transitions from STA11B - await SACTIrsp}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,AB_NR) OR
             idSPDU(tsdu,AB_R)  AND NOT p(2)

      TO STA01 {idle, no TC}

      BEGIN
        Abort(FALSE);
      END;

    PROVIDED idSPDU(tsdu,AB_R) AND p(2)

      TO STA01C {idle, TC con}

      BEGIN
        Abort(TRUE);
      END;

    PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

      TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
{SS-user events}

WHEN SCEP.SACTIrsp

  TO STA713 {data transfer}

  VAR tsdu : TSDUTYPE;

  BEGIN
    BuildAIA(tsdu);

    OUTPUT TCEP.TDTreq(tsdu);

    SpAc(30);
    END;


WHEN SCEP.SUABreq(SSuserData)

  PROVIDED NOT p(2)

    TO STA16 {await TDISind}

    BEGIN
      UserAbort(FALSE,SSuserData);
    END;


  PROVIDED p(2)

    TO STA01A {await AA}

    BEGIN
      UserAbort(TRUE,SSuserData);
    END;


{TS-provider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
    OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
  END;
```

```
FROM STA11C {transitions from STA11C - await SACTDrsp}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,AB_NR) OR
             idSPDU(tsdu,AB_R)  AND NOT p(2)

      TO STA01 {idle, no TC}

      BEGIN
        Abort(FALSE);
      END;

    PROVIDED idSPDU(tsdu,AB_R) AND p(2)

      TO STA01C {idle, TC con}

      BEGIN
        Abort(TRUE);
      END;

    PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

      TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
{SS-user events}

WHEN SCEP.SACTDrsp

  TO STA713 {data transfer}

  VAR tsdu : TSDUTYPE;

  BEGIN
    BuildADA(tsdu);

    OUTPUT TCEP.TDTreq(tsdu);

    SpAc(30);
    END;


WHEN SCEP.SUABreq(SSuserData)

  PROVIDED NOT p(2)

    TO STA16 {await TDISind}

    BEGIN
      UserAbort(FALSE,SSuserData);
    END;


  PROVIDED p(2)

    TO STA01A {await AA}

    BEGIN
      UserAbort(TRUE,SSuserData);
    END;


{TS-provider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData);

  TO STA01 {idle, no TC}

  BEGIN
    OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
  END;
```

```
FROM STA16 {transitions from STA16 - await TDISind}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,DT)  OR
             idSPDU(tsdu,PT)  OR
             idSPDU(tsdu,FN)  OR
             idSPDU(tsdu,DN)  OR
             idSPDU(tsdu,RF)  OR
             idSPDU(tsdu,AC)  OR
             idSPDU(tsdu,GTC) OR
             idSPDU(tsdu,GTA) OR
             idSPDU(tsdu,AI)  OR
             idSPDU(tsdu,AIA) OR
             idSPDU(tsdu,AR)  OR
             idSPDU(tsdu,AE)  OR
             idSPDU(tsdu,AEA) OR
             idSPDU(tsdu,AS)  OR
             idSPDU(tsdu,ED)  OR
             idSPDU(tsdu,MIP) OR
             idSPDU(tsdu,MIA) OR
             idSPDU(tsdu,AD)  OR
             idSPDU(tsdu,ADA)

      TO SAME

      BEGIN
      END;


    PROVIDED idSPDU(tsdu,CN) OR
             idSPDU(tsdu,AA) OR
             idSPDU(tsdu,AB)

      TO STA01 {idle, no TC}

      VAR TSuserData : Bytes64TYPE;

      BEGIN
        TSuserData.l := 0;

        OUTPUT TCEP.TDISreq(TSuserData);

        SpAc(3); {stop timer TIM}
      END;
```

```
  PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

    TO SAME

    BEGIN
      ProtocolErrorAbort;
    END;


{TS-provider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
    SpAc(3); {stop timer TIM}
  END;


{timer events}

WHEN TIMSAP.TIMEOUT

  TO STA01 {idle, no TC}

  VAR TSuserData : Bytes64TYPE;

  BEGIN
    TSuserData.l := 0;

    OUTPUT TCEP.TDISreq(TSuserData);
  END;
```

```
FROM STA18 {transitions from STA18 - await GTA}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,PT) AND p53(TokensVal(tsdu))

      TO SAME

      VAR TokenItem  : TokenSetTYPE;
          SSuserData : Bytes512TYPE;

      BEGIN
        StripPT(tsdu,
                TokenItem,
                SSuserData);

        OUTPUT SCEP.SPTind(TokenItem,
                           SSuserData);
      END;


    PROVIDED idSPDU(tsdu,GTA)

      TO STA713 {data transfer}

      BEGIN
      END;


    PROVIDED idSPDU(tsdu,AB_NR) OR
             idSPDU(tsdu,AB_R)  AND NOT p(2)

      TO STA01 {idle, no TC}

      BEGIN
        Abort(FALSE);
      END;


    PROVIDED idSPDU(tsdu,AB_R) AND p(2)

      TO STA01C {idle, TC con}

      BEGIN
        Abort(TRUE);
      END;
```

```
  PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;

{SS-user events}

WHEN SCEP.SUABreq(SSuserData)

  PROVIDED NOT p(2)

    TO STA16 {await TDISind}

    BEGIN
      UserAbort(FALSE,SSuserData);
    END;

  PROVIDED p(2)

    TO STA01A {await AA}

    BEGIN
      UserAbort(TRUE,SSuserData);
    END;

{TS-provider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
    OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
  END;
```

```
FROM STA19 {await recovery request or SPDU}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,DT)                                    OR
             idSPDU(tsdu,PT)    AND p53(TokensVal(tsdu)) OR
             idSPDU(tsdu,FN_NR) AND p(68)                    OR
             idSPDU(TSDU,FN_R)  AND p(68) AND NOT p(1) AND p(16)

      TO SAME

      BEGIN
      END;


    PROVIDED idSPDU(tsdu,AB_NR) OR
             idSPDU(tsdu,AB_R)  AND NOT p(2)

      TO STA01 {idle, no TC}

      BEGIN
        Abort(FALSE);
      END;


    PROVIDED idSPDU(tsdu,AB_R) AND p(2)

      TO STA01C {idle, TC con}

      BEGIN
        Abort(TRUE);
      END;


    PROVIDED idSPDU(tsdu,AI) AND p(38) AND p(40)

      TO STA11B {await SACTIrsp}

      VAR ReasonCode : ReasonCodeTYPE;

      BEGIN
        StripAI(tsdu,
                ReasonCode);

        OUTPUT SCEP.SACTIind(MapErActInd(ReasonCode.Reason));
      END;
```

```
PROVIDED idSPDU(tsdu,AE) AND p(72) AND p19(SpsnVal(tsdu))

  TO SAME

  BEGIN
    SpAc(31);
  END;


PROVIDED idSPDU(tsdu,MIP) AND p(14) AND p19(SpsnVal(tsdu))

  TO SAME

  BEGIN
    SpAc(23);
  END;


PROVIDED idSPDU(tsdu,AD) AND p(38) AND p(40)

  TO STA11C {await SACTDrsp}

  VAR ReasonCode : ReasonCodeTYPE;

  BEGIN
    StripAD(tsdu,
            ReasonCode);

    OUTPUT SCEP.SACTDind(MapErActInd(ReasonCode.Reason));
  END;


PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

  TO STA16 {await TDISind}

  BEGIN
    ProtocolErrorAbort;
  END;
```

```
{SS-user events}

WHEN SCEP.SUABreq(SSuserData)

  PROVIDED NOT p(2)

    TO STA16 {await TDISind}

    BEGIN
      UserAbort(FALSE,SSuserData);
    END;


  PROVIDED p(2)

    TO STA01A {await AA}

    BEGIN
      UserAbort(TRUE,SSuserData);
    END;

{TS-provider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
    OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
  END;
```

```
FROM STA20 {await recovery SPDU or request}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,PT)  AND p53(TokensVal(tsdu)) OR
             idSPDU(tsdu,AEA) AND p20(SpsnVal(tsdu))   OR
             idSPDU(tsdu,MIA) AND p(17) AND p21(SpsnVal(tsdu))

      TO SAME

      BEGIN
      END;


    PROVIDED idSPDU(tsdu,AB_NR) OR
             idSPDU(tsdu,AB_R)  AND NOT p(2)

      TO STA01 {idle, no TC}

      BEGIN
        Abort(FALSE);
      END;


    PROVIDED idSPDU(tsdu,AB_R) AND p(2)

      TO STA01C {idle, TC con}

      BEGIN
        Abort(TRUE);
      END;


    PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

      TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
{SS-user events}

WHEN SCEP.SACTIreq(Reason)

  PROVIDED p(34) AND p(11)

    TO STA05B {await AIA}

    VAR ReasonCode : ReasonCodeTYPE;
        tsdu       : TSDUTYPE;

    BEGIN
      ReasonCode.Data.1 := 0;
      ReasonCode.Reason := MapErActReq(Reason);

      BuildAI(tsdu,
              ReasonCode);

      OUTPUT TCEP.TDTreq(tsdu);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;


WHEN SCEP.SACTDreq(Reason)

  PROVIDED p(34) AND p(11)

    TO STA05C {await ADA}

    VAR ReasonCode : ReasonCodeTYPE;
        tsdu       : TSDUTYPE;

    BEGIN
      ReasonCode.Data.1 := 0;
      ReasonCode.Reason := MapErActReq(Reason);

      BuildAD(tsdu,
              ReasonCode);

      OUTPUT TCEP.TDTreq(tsdu);
    END;
```

```
    PROVIDED OTHERWISE

      TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;


  WHEN SCEP.SUABreq(SSuserData)

    PROVIDED NOT p(2)

      TO STA16 {await TDISind}

      BEGIN
        UserAbort(FALSE,SSuserData);
      END;


    PROVIDED p(2)

      TO STA01A {await AA}

      BEGIN
        UserAbort(TRUE,SSuserData);
      END;


  {TS-provider events}

  WHEN TCEP.TDISind(Reason,
                    TSuserData)

    TO STA01 {idle, no TC}

    BEGIN
      OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
    END;
```

```
FROM STA713 {data transfer}

  {SPDU events}

  WHEN TCEP.TDTind(tsdu)

    PROVIDED idSPDU(tsdu,DT) AND p(5)

      TO STA713 {data transfer}

      {
       This routine is responsible for REASSEMBLING
       SEGMENTED data SSDUs from incoming DT SPDUs.
      }

      VAR EnclosureItem : EnclosureItemTYPE;
          UserInfo      : SSDUTYPE;

      BEGIN
        StripDT(tsdu,
                EnclosureItem,
                UserInfo);

        IF (EnclosureItem = BEGIN_END)     OR
           (EnclosureItem = BEGIN_NOT_END)
        THEN
          AssembleSSDU.l := 0; {start of new SSDU}

        AppendSSDU(AssembleSSDU,UserInfo);

        IF (EnclosureItem = BEGIN_END)     OR
           (EnclosureItem = NOT_BEGIN_END)
        THEN
         OUTPUT SCEP.SDTind(AssembleSSDU);
      END;


    PROVIDED idSPDU(tsdu,PT) AND p53(TokensVal(tsdu))

      TO STA713 {data transfer}

      VAR TokenItem  : TokenSetTYPE;
          SSuserData : Bytes512TYPE;

      BEGIN
        StripPT(tsdu,
                TokenItem,
                SSuserData);

        OUTPUT SCEP.SPTind(TokenItem,
                           SSuserData);
      END;
```

```
PROVIDED idSPDU(tsdu,FN_NR) AND p(68)

  TO STA09 {await SRELrsp}

  VAR TCdis      : TCdisTYPE;
      SSuserData : Bytes512TYPE;

  BEGIN
    StripFN(tsdu,
            TCdis,
            SSuserData);

    OUTPUT SCEP.SRELind(SSuserData);

    SpAc(8);
  END;


PROVIDED idSPDU(tsdu,FN_R) AND p(68) AND NOT p(1) AND p(16)

  TO STA09 {await SRELrsp}

  VAR TCdis      : TCdisTYPE;
      SSuserData : Bytes512TYPE;

  BEGIN
    StripFN(tsdu,
            TCdis,
            SSuserData);

    OUTPUT SCEP.SRELind(SSuserData);

    SpAc9(TCdis.TCkept);
  END;


PROVIDED idSPDU(tsdu,GTC) AND p(62)

  TO STA713 {data transfer}

  VAR tsdu : TSDUTYPE;

  BEGIN
    BuildGTA(tsdu);

    OUTPUT TCEP.TDTreq(tsdu);

    OUTPUT SCEP.SCGind;

    SpAc11(AvailableTokens);
  END;
```

```
PROVIDED idSPDU(tsdu,AB_NR) OR
         idSPDU(tsdu,AB_R)  AND NOT p(2)

  TO STA01 {idle, no TC}

  BEGIN
    Abort(FALSE);
  END;


PROVIDED idSPDU(tsdu,AB_R) AND p(2)

  TO STA01C {idle, TC con}

  BEGIN
    Abort(TRUE);
  END;


PROVIDED idSPDU(tsdu,AI) AND p(38) AND p(40)

  TO STA11B {await SACTIrsp}

  VAR ReasonCode : ReasonCodeTYPE;

  BEGIN
    StripAI(tsdu,
            ReasonCode);

    OUTPUT SCEP.SACTIind(MapErActInd(ReasonCode.Reason));
  END;
```

```
PROVIDED idSPDU(tsdu,AS) AND p(44)

  TO STA713 {data transfer}

  VAR ActivityId : Bytes6TYPE;
      SSuserData : Bytes512TYPE;

  BEGIN
    StripAS(tsdu,
            ActivityId,
            SSuserData);

    OUTPUT SCEP.SACTSind(ActivityId,
                         SSuserData);

    SpAc(12);
    SpAc(26);
  END;


PROVIDED idSPDU(tsdu,ED) AND p(51)

  TO STA20 {await recovery SPDU or request}

  VAR ReasonCode : ReasonCodeTYPE;
      SSuserData : Bytes512TYPE;

  BEGIN
    StripED(tsdu,
            ReasonCode,
            SSuserData);

    OUTPUT SCEP.SUERind(MapErActInd(ReasonCode.Reason),
                        SSuserData);
  END;


PROVIDED idSPDU(tsdu,MIP) AND p(14) AND p19(SpsnVal(tsdu))

  TO STA713 {data transfer}

  VAR SyncTypeItem : SyncTypeTYPE;
      spsn         : INTEGER;
      SSuserData   : Bytes512TYPE;

  BEGIN
    StripMIP(tsdu,
             SyncTypeItem,
             spsn,
             SSuserData);

    OUTPUT SCEP.SSYNmind(SyncTypeItem,
                         spsn,
                         SSuserData);
    SpAc(23);
  END;
```

```
PROVIDED idSPDU(tsdu,MIA) AND p(17) AND p21(SpsnVal(tsdu))

  TO STA713 {data transfer}

  VAR spsn       : INTEGER;
      SSuserData : Bytes512TYPE;

  BEGIN
    StripMIA(tsdu,
             spsn,
             SSuserData);

    OUTPUT SCEP.SSYNmcnf(spsn,
                         SSuserData);

    SpAc25(spsn);
  END;


PROVIDED idSPDU(tsdu,AD) AND p(38) AND p(40)

  TO STA11C {await SACTDrsp}

  VAR ReasonCode : ReasonCodeTYPE;

  BEGIN
    StripAD(tsdu,
            ReasonCode);

    OUTPUT SCEP.SACTDind(MapErActInd(ReasonCode.Reason));
  END;


PROVIDED OTHERWISE {no predicates are true OR illegal SPDU}

  TO STA16 {await TDISind}

  BEGIN
    ProtocolErrorAbort;
  END;
```

```
{SS-user events}

WHEN SCEP.SDTreq(ssdu)

  PROVIDED p(3)

    {
     This routine is responsible for SEGMENTING
     outgoing data SSDUs, if required.
    }

    VAR i           : INTEGER:
        index       : INTEGER;  {points to last SSDU byte sent}
        remainder   : INTEGER;  {no. of SSDU bytes still to send}
        MaxInfoLen  : INTEGER;  {maximum User Info field length}
        InfoLen     : INTEGER;  {actual  User Info field length}
        SSDUbegin,  : BOOLEAN;  {begin of SSDU indicator}
        SSDUend     : BOOLEAN;  {end   of SSDU indicator}
        UserInfo    : SSDUTYPE; {User Info field}
        tsdu        : TSDUTYPE;

        EnclosureItem : EnclosureItemTYPE;

    BEGIN
      IF MaxTSDU0 = 0
      THEN              {do not segment the SSDU}

        BEGIN
          BuildDT(tsdu,
                  BEGIN_END, {Enclosure Item}
                  ssdu);

          OUTPUT TCEP.TDTreq(tsdu);
        END;
```

```
ELSE            {segment the SSDU}

  BEGIN
    index      := 0;
    remainder  := ssdu.l;
    SSDUbegin  := TRUE;
    SSDUend    := FALSE;
    MaxInfoLen := MaxTSDU0 - DT_HEADER_LEN - 3;

    {Where the '3' is the length of the GT SPDU which
     will precede DT in the TSDU for basic concatenation.}

    WHILE NOT SSDUend DO
      BEGIN

        IF remainder > MaxInfoLen
        THEN
          BEGIN
            InfoLen := MaxInfoLen;
            SSDUend := FALSE;
          END;
        ELSE
          BEGIN
            InfoLen := remainder;
            SSDUend := TRUE;
          END;

        IF InfoLen > 0
        THEN
          FOR i := 1 TO InfoLen DO
            UserInfo.d[i] := ssdu.d[index+i];
        UserInfo.l := InfoLen;

        {Set EnclosureItem}

        IF NOT SSDUbegin AND NOT SSDUend
        THEN
          EnclosureItem := NOT_BEGIN_NOT_END;

        IF NOT SSDUbegin AND SSDUend
        THEN
          EnclosureItem := NOT_BEGIN_END;

        IF SSDUbegin AND NOT SSDUend
        THEN
          EnclosureItem := BEGIN_NOT_END;

        IF SSDUbegin AND SSDUend
        THEN
          EnclosureItem := BEGIN_END;

        BuildDT(tsdu,
                EnclosureItem,
                UserInfo);
```

```
            OUTPUT TCEP.TDTreq(tsdu);

            index     := index     + InfoLen;
            remainder := remainder - InfoLen;
            SSDUbegin := FALSE;
          END;
      END;
  END;


PROVIDED OTHERWISE

  TO STA16 {await TDISind}

  BEGIN
    ProtocolErrorAbort;
  END;
```

```
WHEN SCEP.SPTreq(Tokens,
                 SSuserData)

  PROVIDED p53(Tokens)

    TO STA713 {data transfer}

    VAR tsdu : TSDUTYPE;

    BEGIN
      BuildPT(tsdu,
              Tokens,
              SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;


WHEN SCEP.SCGreq

  PROVIDED p(55)

    TO STA18 {await GTA}

    VAR tsdu : TSDUTYPE;

    BEGIN
      BuildGTC(tsdu);

      OUTPUT TCEP.TDTreq(tsdu);

      SpAc11([]);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;
```

```
WHEN SCEP.SSYNmreq(SyncType,
                   spsn,
                   SSuserData)

  PROVIDED p(15)

    TO STA713 {data transfer}

    VAR tsdu : TSDUTYPE;

    BEGIN
      BuildMIP(tsdu,
               SyncType,
               spsn,
               SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);

      SpAc(24);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;


WHEN SCEP.SSYNmrsp(spsn,
                   SSuserData)

  PROVIDED p(18) AND p21(spsn)

    TO STA713 {data transfer}

    VAR tsdu : TSDUTYPE;

    BEGIN
      BuildMIA(tsdu,
               spsn,
               SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);

      SpAc25(spsn);
    END;
```

```
  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;


WHEN SCEP.SUERreq(Reason,
                  SSuserData)

  PROVIDED p(50)

    TO STA19 {await recovery request or SPDU}

    VAR ReasonCode : ReasonCodeTYPE;
        tsdu       : TSDUTYPE;

    BEGIN
      ReasonCode.Data.l := 0;
      ReasonCode.Reason := MapErActReq(Reason);

      BuildED(tsdu,
              ReasonCode,
              SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;
```

```
WHEN SCEP.SACTRreq(ActivityId,
                   OldActivityId,
                   spsn,
                   CallingSSuserRef,
                   CalledSSuserRef,
                   CommonRef,
                   AdditionalRef,
                   SSuserData)

  PROVIDED p(45)

    TO STA713 {data transfer}

    VAR tsdu : TSDUTYPE;

    BEGIN
      BuildAR(tsdu,
              CalledSSuserRef,
              CallingSSuserRef,
              CommonRef,
              AdditionalRef,
              OldActivityId,
              spsn,
              ActivityId,
              SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);

      SpAc(12);
      SpAc27(spsn);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;
```

```
WHEN SCEP.SACTIreq(Reason)

  PROVIDED p(34) AND p(39)

    TO STA05B {await AIA}

    VAR ReasonCode : ReasonCodeTYPE;
        tsdu       : TSDUTYPE;

    BEGIN
      ReasonCode.Data.l := 0;
      ReasonCode.Reason := MapErActReq(Reason);

      BuildAI(tsdu,
              ReasonCode);

      OUTPUT TCEP.TDTreq(tsdu);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;


WHEN SCEP.SACTDreq(Reason)

  PROVIDED p(34) AND p(39)

    TO STA05C {await ADA}

    VAR ReasonCode : ReasonCodeTYPE;
        tsdu       : TSDUTYPE;

    BEGIN
      ReasonCode.Data.l := 0;
      ReasonCode.Reason := MapErActReq(Reason);

      BuildAD(tsdu,
              ReasonCode);

      OUTPUT TCEP.TDTreq(tsdu);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;
```

```
WHEN SCEP.SACTEreq(spsn,
                   SSuserData)

  PROVIDED p(71)

    TO STA04B {await AEA}

    VAR tsdu : TSDUTYPE;

    BEGIN
      BuildAE(tsdu,
              spsn,
              SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);

      SpAc(13);
      SpAc(24);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;


WHEN SCEP.SRELreq(SSuserData)

  PROVIDED p(63) AND NOT p(64)

    TO STA03 {await DN}

    VAR TCdis : TCdisTYPE;
        tsdu  : TSDUTYPE;

    BEGIN
      TCdis.TCkept   := FALSE;     {for FNnr}
      TCdis.ABreason := NO_ABORT; {No Abort in progress}

      BuildFN(tsdu,
              TCdis,
              SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);

      SpAc(8);
    END;
```

```
  PROVIDED p(63) AND p(64)

    TO STA03 {await DN}

    VAR TCdis : TCdisTYPE;
        tsdu  : TSDUTYPE;

    BEGIN
      TCdis.TCkept   := TRUE;      {for FNr}
      TCdis.ABreason := NO_ABORT; {No Abort in progress}

      BuildFN(tsdu,
              TCdis,
              SSuserData);

      OUTPUT TCEP.TDTreq(tsdu);

      SpAc(7);
    END;


  PROVIDED OTHERWISE

    TO STA16 {await TDISind}

    BEGIN
      ProtocolErrorAbort;
    END;


WHEN SCEP.SUABreq(SSuserData)

  PROVIDED NOT p(2)

    TO STA16 {await TDISind}

    BEGIN
      UserAbort(FALSE,SSuserData);
    END;


  PROVIDED p(2)

    TO STA01A {await AA}

    BEGIN
      UserAbort(TRUE,SSuserData);
    END;
```

```
{TS-provider events}

WHEN TCEP.TDISind(Reason,
                  TSuserData)

  TO STA01 {idle, no TC}

  BEGIN
    OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
  END;
```

{
Part 2:

Transitions for handling invalid intersections between SPM states and the following incoming events:

i. SS-user events
ii. TS-provider events.
}

```
WHEN SCEP.SCONreq

  FROM STA01A,STA01B,STA02A,STA03 ,STA04B,STA05B,STA05C,STA08 ,
       STA09 ,STA10B,STA11B,STA11C,STA16 ,STA18 ,STA19 ,STA20 ,
       STA713

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;


WHEN SCEP.SCONrsp

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA09 ,STA10B,STA11B,STA11C,STA16 ,STA18 ,STA19 ,STA20 ,
       STA713

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;


WHEN SCEP.SDTreq

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA08 ,STA09 ,STA10B,STA11B,STA11C,STA16 ,STA18 ,STA19 ,
       STA20

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
WHEN SCEP.SPTreq

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA08 ,STA11B,STA11C,STA16 ,STA18 ,STA19 ,STA20

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;


WHEN SCEP.SCGreq

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA08 ,STA09 ,STA10B,STA11B,STA11C,STA16 ,STA18 ,STA19 ,
       STA20

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
WHEN SCEP.SSYNmreq

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA08 ,STA09 ,STA10B,STA11B,STA11C,STA16 ,STA18 ,STA19 ,
       STA20

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;


WHEN SCEP.SSYNmrsp

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA08 ,STA11B,STA11C,STA16 ,STA18 ,STA19 ,STA20

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
WHEN SCEP.SUERreq

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA08 ,STA11B,STA11C,STA16 ,STA18 ,STA19 ,STA20

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;


WHEN SCEP.SACTSreq

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA08 ,STA09 ,STA10B,STA11B,STA11C,STA16 ,STA18 ,STA19 ,
       STA20

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
WHEN SCEP.SACTRreq

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA08 ,STA09 ,STA10B,STA11B,STA11C,STA16 ,STA18 ,STA19 ,
       STA20

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;


WHEN SCEP.SACTIreq

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA05B,STA05C,STA08 ,
       STA09 ,STA10B,STA11B,STA11C,STA16 ,STA18 ,STA19

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
WHEN SCEP.SACTIrsp

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA08 ,STA09 ,STA10B,STA11C,STA16 ,STA18 ,STA19 ,STA20 ,
       STA713

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;


WHEN SCEP.SACTDreq

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA05B,STA05C,STA08 ,
       STA09 ,STA10B,STA11B,STA11C,STA16 ,STA18 ,STA19

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
WHEN SCEP.SACTDrsp

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA08 ,STA09 ,STA10B,STA11B,STA16 ,STA18 ,STA19 ,STA20 ,
       STA713

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;


WHEN SCEP.SACTEreq

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA08 ,STA09 ,STA10B,STA11B,STA11C,STA16 ,STA18 ,STA19 ,
       STA20

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
WHEN SCEP.SACTErsp

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA08 ,STA09 ,STA11B,STA11C,STA16 ,STA18 ,STA19 ,STA20 ,
       STA713

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;


WHEN SCEP.SRELreq

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA08 ,STA09 ,STA10B,STA11B,STA11C,STA16 ,STA18 ,STA19 ,
       STA20

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
WHEN SCEP.SRELrsp

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA08 ,STA10B,STA11B,STA11C,STA16 ,STA18 ,STA19 ,STA20 ,
       STA713

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;


WHEN SCEP.SUABreq

  FROM STA01

    TO SAME

      BEGIN
        OUTPUT SCEP.SPABind(PROTOCOL_ERROR);
      END;

  FROM STA01A,STA01C,STA16

    TO STA16 {await TDISind}

      BEGIN
        ProtocolErrorAbort;
      END;
```

```
WHEN TCEP.TCONind

  FROM STA01A,STA01B,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,
       STA08 ,STA09 ,STA10B,STA11B,STA11C,STA16 ,STA18 ,STA19 ,
       STA20 ,STA713

    TO STA01 {idle, no TC}

      VAR TSuserData : Bytes64TYPE;

      BEGIN
        TSuserData. := 0;
        OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
        OUTPUT TCEP.TDISreq(TSuserData);
      END;


WHEN TCEP.TCONcnf

  FROM STA01

    TO SAME

      VAR TSuserData : Bytes64TYPE;

      BEGIN
        TSuserData.1 := 0;
        OUTPUT TCEP.TDISreq(TSuserData);
      END;

  FROM STA01A,STA01C,STA02A,STA03 ,STA04B,STA05B,STA05C,STA08 ,
       STA09 ,STA10B,STA11B,STA11C,STA16 ,STA18 ,STA19 ,STA20 ,
       STA713

    TO STA01 {idle, no TC}

      VAR TSuserData : Bytes64TYPE;

      BEGIN
        TSuserData.1 := 0;
        OUTPUT SCEP.SPABind(TRANSPORT_DISCONNECT);
        OUTPUT TCEP.TDISreq(TSuserData);
      END;


WHEN TCEP.TDISind

  FROM STA01

    TO SAME

      BEGIN
      END;
```

APPENDIX D. The Software Development System

Hardware:

Processor:		Intel 80386
Clock Frequency:		25 MHz
Memory:	RAM:	4 Mb
	drive A:	1.2 Mb high density floppy drive
	drive B:	360 kb floppy drive
	drive C:	90 Mb hard drive, Western Digital controller
Monitor:		EGA
Power Supply:		220 VAC 50 Hz 300 W uninterruptable

Software:

Operating system:	Interactive AT&T UNIX System V/386 Release 3.2
	featuring: C Software Development System
	VP/ix: MS-DOS/UNIX integration
Text editor:	Multi-Edit (MS-DOS)

APPENDIX E. Session Entity Source File Listings

This appendix lists the following session entity source files:

E.1 Session entity configuration header file listing: sconfig.h

```
/*
 * SESSION ENTITY CONFIGURATION HEADER FILE: sconfig.h
 * ED van der Westhuizen   September 1989
 *
 * Session Entity configuration constants.
 *
 * prerequisite header files:
 *   osdeps.h     for:  TRUE, FALSE.
 *   transport.h  for:  qos_type.
 *   session.h    for:  HDX, MIS, ACT, EXC.
 *   system.h     for:  CLOCK.
 */

#ifndef __SCONFIG__  /* avoid multiple inclusion */
#define __SCONFIG__

#define FU_SUP       (HDX|MIS|ACT|EXC)  /* supported functional units */
#define FASTTIMER    (60000/CLOCK)      /* abort   timer value: 60 secs */
#define SLOWTIMER    (10000/CLOCK)      /* connect timer value: 10 secs */
#define REUSE_TC     TRUE               /* reuse TC option */
#define NSPMS        16                 /* maximum simultaneous SCs */
#define NSSAPS       1                  /* maximum SSAPs */
#define TEXP_LOCAL   FALSE              /* transport expedited data option */
#define VERSION      1                  /* session protocol version number */
#define PROTOCOL     0                  /* session protocol options: */
                                        /* 1 = extended concatenation    */
                                        /* 0 = no extended concatenation */

static qos_type defaultQOSS;            /* default QOSS values */
static qos_type defaultQOTS;            /* default QOTS values */

#endif  /* __SCONFIG__ */
```

E.2 Transport entity header file listing: trans.h

```
/*
 * TRANSPORT ENTITY HEADER FILE: trans.h
 * ED van der Westhuizen   September 1989
 *
 * External declarations of Transport Entity functions.
 *
 * prerequisite header files:
 *   osdeps.h     for:  boolean, uint8, pointer.
 *   address.h    for:  tsap_selector, nsap_address.
 *   transport.h  for:  qos_type.
 *   session.h    for:  struct buf.
 */

#ifndef __TRANS__  /* avoid multiple inclusion */
#define __TRANS__

#ifdef LINT_ARGS

extern void TSUadd(tsap_selector *,     /* local TSAP selector */
                   int (*)(),           /* user TCONind handler */
                   int (*)(),           /* user TCONcnf handler */
                   int (*)(),           /* user TDISind handler */
                   int (*)());          /* user TDTind  handler */

extern pointer UCONreq(tsap_selector *, /* calling TSAP selector */
                       nsap_address *,  /* called  NSAP address */
                       tsap_selector *, /* called  TSAP selector */
                       qos_type *,      /* proposed QOTS */
                       boolean,         /* proposed TEXP option */
                       uint8 *);        /* TS-user data */

extern void UCONres(pointer,            /* TCEP identifier */
                    qos_type *,         /* selected QOTS */
                    boolean,            /* selected TEXP option */
                    uint8 *);           /* TS-user data */

extern void UDISreq(pointer,            /* TCEP identifier */
                    uint8 *);           /* TS-user data */

extern void UDATreq(pointer,            /* TCEP identifier */
                    struct buf *,       /* TSDU buffer */
                    boolean);           /* end of TSDU flag */

extern void do_transport_queue(void);

#else

extern void    TSUadd();
extern pointer UCONreq();
extern void    UCONres();
extern void    UDISreq();
extern void    UDATreq();
extern void    do_transport_queue();

#endif  /* LINT_ARGS */

#endif  /* __TRANS__ */
```

E.3 Buffer manager header file listing: buffer.h

```
/*
 * UPPER LAYER BUFFER MANAGER HEADER FILE: buffer.h
 * ED van der Westhuizen   September 1989
 *
 * External declarations of buffer manager functions.
 *
 * prerequisite header files:
 *   session.h  for:  struct buf.
 */

#ifndef __BUFFER__  /* avoid multiple inclusion */
#define __BUFFER__

#ifdef LINT_ARGS

   extern int        init_buffers(int,int);
   extern struct buf *balloc(int);
   extern void       bclear(struct buf *);
   extern void       bfree(struct buf *);

#else

   extern int        init_buffers();
   extern struct buf *balloc();
   extern void       bclear();
   extern void       bfree();

#endif  /* LINT_ARGS */

#endif  /* __BUFFER__ */
```

E.4 Buffer manager source file listing: buffer.c

```
/*
 * UPPER LAYER BUFFER MANAGER SOURCE FILE: buffer.c
 * ED van der Westhuizen   September 1989
 *
 * external visibility:
 *   int        init_buffers()
 *   struct buf *balloc(int)
 *   void       bclear(struct buf *)
 *   void       bfree(struct buf *)
 */

#include "osdeps.h"    /* for: TRUE, FALSE, NULL. */
#include "address.h"   /* needed by session.h */
#include "transport.h" /* needed by session.h */
#include "session.h"   /* for: struct buf. */
#include "queue.h"     /* queue manager external declarations */

/*
 * Buffer Pool Head variable
 */

static struct {
            struct buf *first;  /* pointer to first element */
            struct buf *last;   /* pointer to last element */
       } bufQhead;

/*
 * FUNCTION:  binit
 *
 *            This function initializes the buffer pool to an empty state.
 *            All previously held buffers are lost. This must be the first
 *            function to be called in the buffer manager initialization
 *            process.
 *
 * INPUTS:    none.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     none.
 */

static void binit()
{
 bufQhead.first = bufQhead.last = (struct buf *) &bufQhead;
}
```

```
/*
 * FUNCTION:  badd
 *
 *            This function adds a given buffer descriptor to the
 *            buffer pool. The .start and .size fields of the buffer
 *            descriptor must be initialized before this function is called.
 *            Before any session layer activity can occur, a reasonable
 *            number of buffers must have been added to the buffer pool.
 *
 * INPUTS:    bp - pointer to the buffer descriptor.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     QInsert.
 */

static void badd(bp)
struct buf *bp;
{
 QInsert(&bufQhead,bp);
}

/*
 * FUNCTION:  init_buffers
 *
 *            This function allocates memory for, and initializes, the
 *            buffer pool. This function must be called once by the SS-user
 *            before any calls to the session entity are made.
 *
 * INPUTS:    nbufs - number of buffers to allocate.
 *            bufsz - individual buffer size.
 *
 * OUTPUTS:   returns: TRUE  if buffer pool set-up was successful,
 *                     FALSE if not.
 *
 * CALLS:     binit, badd, os_malloc, QInit.
 */

int init_buffers(nbufs,bufsz)
int nbufs;
int bufsz;
{
 int           i;
 struct buf    *bp;          /* pointer to buffer descriptor */
 unsigned char *start;       /* pointer to start of buffer */
 binit();                    /* initialize buffer pool to empty state */
 for (i = 0; i < nbufs; i++) {
     bp = (struct buf *) os_malloc(sizeof(struct buf));
     start = (unsigned char *) os_malloc(bufsz);
     if (bp && start) {
        QInit(bp);           /* initialize buffer queue element */
        bp->start  = start;  /* pointer to start of buffer */
        bp->addr   = NULL;   /* pointer to first data byte */
        bp->length = 0;      /* length of data */
        bp->size   = bufsz;  /* size of buffer */
        badd(bp);            /* add buffer descriptor to buffer pool */
     }
     else
        return(FALSE);       /* memory could not be allocated for pool */
 }
 return(TRUE);
}
```

```
/*
 * FUNCTION:  balloc
 *
 *            This function allocates a buffer from the buffer pool (if one
 *            is available) and initializes its .length and .addr fields
 *            to create a buffer of length n. The eosdu field is
 *            initialized to TRUE.
 *
 * INPUTS:    n - length of buffer to allocate.
 *                If n is negative, the length of the returned buffer is
 *                set to the maximum possible. This is used by the session
 *                entity when it needs to allocate a new receive buffer
 *                and does not know the length of the next incoming SSDU.
 *
 * OUTPUTS:   returns: a pointer to the allocated buffer's descriptor
 *                     if one was available,
 *                     NULL if not.
 *
 * CALLS:     QRemove.
 */

struct buf *balloc(n)
int n;
{
 struct buf *bp;          /* pointer to buffer descriptor */

 if ((bp = bufQhead.last) != (struct buf*) &bufQhead) {
    QRemove(bp);          /* remove buffer from pool */
    if (n >= 0) {
       bp->length = n;
       bp->addr   = bp->start + bp->size - n;
    }
    else {
       bp->length = bp->size;
       bp->addr   = bp->start;
    }
    bp->eosdu = TRUE;
    return(bp);           /* buffer successfully allocated */
 }
 return(NULL);            /* the buffer pool is empty */
}

/*
 * FUNCTION:  bclear
 *
 *            This function resets a given buffer descriptor's
 *            .addr and .length fields to indicate a buffer of
 *            zero length.
 *
 * INPUTS:    bp - pointer to buffer descriptor.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     none.
 */

void bclear(bp)
struct buf *bp;
{
 bp->length = 0;
 bp->addr   = bp->start + bp->size;
}
```

```
/*
 * FUNCTION:  bfree
 *
 *            This function returns a previously allocated buffer
 *            to the buffer pool.
 *
 * INPUTS:    bp - pointer to buffer descriptor.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     QInsert.
 */

void bfree(bp)
struct buf *bp;
{
 QInsert(&bufQhead,bp);
}
```

E.5 Session entity source file listing: session.c

```
/*
 * SESSION ENTITY SOURCE FILE: session.c
 * ED van der Westhuizen   September 1989
 *
 * This source file implements one Session Entity for supporting one RTS.
 *
 * External visibility:
 *
 * Functions called by SS-user:       Functions called by TS-provider:
 *
 *   void            init_session()     int TCONind()
 *   int             s_activate()       int TCONcnf()
 *   int             s_deactivate()     int TDISind()
 *   struct Smachine *s_connect()       int TDTind()
 *   int             session()
 *   void            do_session_queue()
 */

/*-------------------------------------------------------------*
 * HEADER FILES                                                *
 *-------------------------------------------------------------*/

#include "osdeps.h"     /* os dependent definitions & macros */
#include "queue.h"      /* queue manager external declarations */
#include "system.h"     /* timer manager definitions */
#include "access.h"     /* PDU parsing and formatting macros */
#include "address.h"    /* address definitions */
#include "transport.h"  /* transport interface definitions */
#include "session.h"    /* session interface definitions */
#include "sconfig.h"    /* session entity configuration constants */
#include "trans.h"      /* transport entity external declarations */
#include "buffer.h"     /* buffer manager external declarations */

/*-------------------------------------------------------------*
 * MACRO DEFINITIONS                                           *
 *-------------------------------------------------------------*/

/* SPM state transition to newstate */

#define TO(newstate) CURNTs->STATE = newstate

/* identifier for same SPM state */

#define SAME CURNTs->STATE

/* return minimum value */

#define min(A,B) ((A) < (B) ? (A) : (B))

/*-------------------------------------------------------------*
 * MANIFEST CONSTANTS                                          *
 *-------------------------------------------------------------*/

/*
 * session timer names
 */

#define CONNECT_TIMER  0  /* connect timer */
#define ABORT_TIMER    1  /* abort   timer */
```

```
/*
 * Transport Event Identifiers for Transport Event Queue
 */

#define TCONIND  0  /* T-CONNECT.indication    */
#define TCONCNF  1  /* T-CONNECT.confirm       */
#define TDTIND   2  /* T-DATA.indication       */
#define TDISIND  3  /* T-DISCONNECT.indication */

/*
 * SPDU identifiers for idSPDU()
 */

#define RF    0  /* REFUSE           */
#define RFr   1  /* REFUSE reuse     */
#define RFnr  2  /* REFUSE not reuse */
#define CN    3  /* CONNECT          */
#define AC    4  /* ACCEPT           */
#define AB    5  /* ABORT            */
#define ABr   6  /* ABORT reuse      */
#define ABnr  7  /* ABORT not reuse  */
#define AA    8  /* ABORT ACCEPT     */

/*
 * values for the 1st byte of the Reason Code parameter for: RF ED AI AD SPDUs
 */

#define SSU_UNSPECIFIED         0   /* RF ED AI AD */
#define SSU_CONGESTED           1   /* RF ED AI AD */
#define SSU_SEE_DATA            2   /* RF          */
#define SEQUENCE_ERROR          3   /*    ED AI AD */
#define LOCAL_SSU_ERROR         5   /*    ED AI AD */
#define PROCEDURE_ERROR         6   /*    ED AI AD */
#define DEMAND_DK             128   /*    ED AI AD */
#define CALLED_SSAP_UNKNOWN   129   /* RF          */
#define CALLED_SSU_UNATTACHED 130   /* RF          */
#define SSP_CONGESTED         131   /* RF          */
#define PROPOSED_PROTOCOL     132   /* RF          */

/*
 * Reason parameter values for SPABind primitive
 */

#define T_DISCONNECT    1  /* Transport Disconnect */
#define PROTOCOL_ERROR  4  /* Protocol Error       */
#define UNDEFINED       8  /* Undefined Error      */

/*
 * ABreason field values for struct TCdis, i.e., values for bits 2-4 of the
 * Transport Disconnect parameter for RF AB FN SPDUs.
 */

#define NO_ABORT        0  /* no abort in progress */
#define USER_ABORT      2  /* user abort           */
#define PROTOCOL_ERROR  4  /* provider abort       */
#define NO_REASON       8  /* provider abort       */

/*
 * Session error identifiers for SessionError()
 */

#define SERBUF  0  /* buffer cannot be allocated */
#define SERMEM  1  /* out of memory              */
#define SERSID  2  /* invalid SCEP identifier    */
#define SERTID  3  /* invalid TCEP identifier    */
```

```
/*-------------------------------------------------------------*
 * DATA TYPE DEFINITIONS                                       *
 *-------------------------------------------------------------*/

/* 9 bytes */

struct Bytes9 {
     uint8 data[9];
     int   len;
};

/* Transport disconnect parameter for RF AB FN SPDUs */

struct TCdis {
     uint8   ABreason;     /* bits 2-4 - AB SPDU only */
     boolean TCkept;       /* bit  1 */
};

/* Transport Layer Event Queue Element Type */

typedef struct TQelement {
             struct TQelement *next;     /* next element */
             struct TQelement *prev;     /* previous element */
             uint8         event;        /* transport event identifier */
             nsap_address  clgNSAPaddr;  /* calling NSAP address */
             nsap_address  cldNSAPaddr;  /* called  NSAP address */
             tsap_selector clgTSAPid;    /* calling TSAP identifier */
             tsap_selector cldTSAPid;    /* called  TSAP identifier */
             pointer       TCEPid;       /* TCEP identifier */
             qos_type      qots;         /* Quality Of Transport Service */
             boolean       texp;         /* transport expedited data option */
             struct buf    *tsdu;        /* TSDU buffer */
             uint8         reason;       /* disconnect reason */
             uint8         *TSUdata;     /* TS-user data */
             boolean       eotsdu;       /* end of TSDU flag */
        } TQelement;

/*-------------------------------------------------------------*
 * EXTERNAL FUNCTION DECLARATIONS                              *
 *-------------------------------------------------------------*/

#ifdef LINT_ARGS  /* only if lint and cc support parameter types */

  /* timer management module */
   extern void init_memory(void);
   extern void init_timers(void);
   extern void newtimer(char *,int,int,int,int,int (*)());
   extern void cantimer(char *,int,int);
   extern void do_timer_queue(void);
   extern void clock(void);

   /* byte string copy */
   extern void bcopy(char *,char *,int);

   /* SAP address comparison facilities */
   extern int nsap_cmp(nsap_address  *,nsap_address  *);
   extern int tsap_cmp(tsap_selector *,tsap_selector *);
   extern int ssap_cmp(ssap_selector *,ssap_selector *);

#else

  /* timer management module */
   extern void init_memory();
   extern void init_timers();
   extern void newtimer();
   extern void cantimer();
   extern void do_timer_queue();
   extern void clock();
```

```
   /* byte string copy */
   extern void bcopy();

   /* SAP address comparison facilities */
   extern int nsap_cmp();
   extern int tsap_cmp();
   extern int ssap_cmp();

#endif

/*-------------------------------------------------------*
 * STATIC VARIABLE DEFINITIONS                           *
 *-------------------------------------------------------*/

/*
 * The array of Session Protocol Machines (SPMs).
 * Each structure holds all relevant information for a particular session
 * connection. A pointer to an SPM structure is a SCEP identifier.
 */

static struct Smachine {

            enum {
               STA01,                  /* idle, no TC   */
               STA01A,                 /* await AA      */
               STA01B,                 /* await TCONcnf */
               STA01C,                 /* idle, TC con  */
               STA02A,                 /* await AC      */
               STA08,                  /* await SCONrsp */
               STA16,                  /* await TDISind */
               STA713                  /* data transfer */
            }           STATE;         /* MAJOR STATE VARIABLE */

                                       /* REQUIRED VARIABLES */
            boolean     Vtca,          /* TC acceptor */
                        Texp,          /* transport expeditied data option */
                        Vsc,           /* SS-user right to issue SSYNmrsp */
                        Vact,          /* activity in progress */
                        Vnextact,      /* next Vact value */
                        Vtrr;          /* reuse of TC allowed */
            uint32      Va,            /* lowest sn for synch point cnf */
                        Vm;            /* next sn */

                                       /* OTHER VARIABLES */
            uint16      fus;           /* selected functional units */
            uint8       AvTokens;      /* available tokens */
            uint8       AsTokens;      /* owned tokens */
            uint8       protocol;      /* remote SPM protocol options */
            uint8       version;       /* selected version number */
            uint16      max0TSDUlen;   /* selected max TSDU len: I to R */
            uint16      max1TSDUlen;   /* selected max TSDU len: R to I */
            qos_type    qoss;          /* selected QOSS */
            qos_type    qots;          /* selected QOTS */
            uint8       TEMPtoken;     /* temporary token assignments */
            ssap_address remSSAPaddr;  /* remote SSAP address */
            pointer     TCEPid;        /* TCEP identifier */
            struct idu  idu;           /* temporary SIDU storage */

      } SPM[NSPMS];

/* Transport Layer Event Queue Head */

static struct {
            TQelement *first;          /* first element */
            TQelement *last;           /* last element */
      } TQhead;
```

```
/* other Session Entity variables */

static void            (*SSuser)();     /* SS-user ind/cnf handler */
static ssap_selector   locSSAPid;       /* local SSAP identifier */
static tsap_selector   locTSAPid;       /* local TSAP identifier */
static struct Smachine *CURNTs;         /* pointer to current SPM (SCEPid) */
static int             nssaps;          /* number of SSAPs registered */
static uint16          MAXTSDULEN;      /* maximum TSDU length */

/*-------------------------------------------------------*
 * static FUNCTION DEFINITIONS                           *
 *-------------------------------------------------------*/

#ifdef DEBUG
#include "debug.c"   /* session debug source */
#endif

#include "sprmtvs.c" /* session   ind/cnf primitive functions */
#include "tprmtvs.c" /* transport req/rsp primitive functions */
#include "funcs1.c"  /* ClockInterrupt, FU, AV, p, SpAc. */
#include "strip.c"   /* SPDU stripping functions */
#include "build.c"   /* SPDU building functions */
#include "funcs2.c"  /* reuseTC, idSPDU, UserAbort, SessionError, get_buf, */
                     /* get_mem. */

/*
 * FUNCTION:  SessionTimeOut
 *
 *            This function is called by do_timer_queue() from
 *            ClockInterrupt() at SIGALRM interrupt time if it
 *            detects a session timer timeout.
 *
 * INPUTS:    s         - the SCEPid  (pointer to SPM).
 *            name      - session timer name.
 *            subscript - unused, no significance.
 *            datum     - unused, no significance.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     TDISreq.
 */

static void SessionTimeOut(s,name,subscript,datum)
register struct Smachine *s;
int                      name;
int                      subscript;
int                      datum;
{
#ifdef DEBUG
  int i = NSPMS;
  while (s != &SPM[--i]) ;
  printf("\nSessionTimeOut(): A timer has expired.\n");
  printf("   SPM:          %d\n",i);
  printf("   From state:   %s\n",Sstate[(int) s->STATE]);
  printf("   Input event:  %s\n",timer[name]);
#endif
```

```
CURNTs = s;
switch (s->STATE) {
case STA01A:                        /*** await ABORT ACCEPT SPDU ***/
case STA16:                         /*** await T-DISCONNECT.indication ***/
     switch (name) {
     case ABORT_TIMER:                   /* abort timer timeout */
          TDISreq(s->TCEPid);               /* release the TC */
          TO(STA01);                        /* idle, no TC */
          break;

     default:                            /* Any other timeouts */
          break;                            /* invalid, so ignore */
     }
     break;


case STA01C:                        /*** idle, TC con ***/
     switch (name) {
     case CONNECT_TIMER:                 /* connect timer timeout */
          TDISreq(s->TCEPid);               /* release the TC */
          TO(STA01);                        /* idle, no TC */
          break;

     default:                            /* Any other timeouts */
          break;                            /* invalid, so ignore */
     }
     break;


default:                            /* Any other state */
     break;                            /* timeouts invalid, so ignore */
}

#ifdef DEBUG
  printf("   To state:     %s\n",Sstate[(int) s->STATE]);
#endif
}
```

```
/*-------------------------------------------------------*
 * EXTERNAL FUNCTION DEFINITIONS                         *
 *-------------------------------------------------------*/

/*
 * FUNCTION:  init_session
 *
 *            The SS-user calls this function to initialize
 *            the session entity. It must be the first session entity
 *            function called and must be called only once.
 *
 *            It initializes: 1. The local maximum TSDU length,
 *                            2. The SPMs,
 *                            3. The transport layer interface,
 *                            4. transport layer event queue,
 *                            5. timer module.
 *
 * INPUTS:    tsap_id - a pointer to the local TSAPid.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     TSUadd, init_memory, init_timers, ClockInterrupt, bcopy.
 */

void init_session(tsap_id)
register tsap_selector *tsap_id;
{
 extern int TCONind();  /* T-CONNECT.indication handler */
 extern int TCONcnf();  /* T-CONNECT.confirm handler */
 extern int TDISind();  /* T-DISCONNECT.indication handler */
 extern int TDTind();   /* T-DATA.indication handler */
 int i;
 struct buf *bp;

 /* determine the local maximum TSDU length (= max session layer buffer size) */
 bp = balloc(0);
 MAXTSDULEN = bp->size;
 bfree(bp);

 /* initialize relevant SPM variables */
 for (i = 0; i < NSPMS; i++) {
     SPM[i].STATE   = STA01;      /* INITIAL STATE = idle, no TC */
     SPM[i].Vtca    = FALSE;      /* not TC acceptor */
 }

 /* Initialise Transport Layer Event Queue Head to EMPTY */
 TQhead.first = TQhead.last = (TQelement *) &TQhead;

 /* store local TSAP id */
 bcopy(tsap_id, &locTSAPid, sizeof(tsap_selector));

 /* SS-user is inactive */
 nssaps = 0;

 /* register this TS-user with the transport entity */
 TSUadd(tsap_id,TCONind,TCONcnf,TDISind,TDTind);

 /* initialize timer module */
 init_memory();     /* initialize timer memory */
 init_timers();     /* initialize timer management module */
 ClockInterrupt();  /* start timer interrupt mechanism */

#ifdef DEBUG
  printf("\ninit_session(): Initialize the session entity.\n");
  printf("   local TSAPid: ");
  printHEX(tsap_id->addr,tsap_id->len);
  printf("   max TSDU length = %d\n",MAXTSDULEN);
#endif
}
```

```
/*
 * FUNCTION:  s_activate
 *
 *            The SS-user calls this function to register itself with
 *            the session entity. The SS-user must call this function
 *            before it can use any session services.
 *
 * INPUTS:    ssap_id - a pointer to the unique local SSAPid which identifies
 *                      the SS-user within the scope of the supporting TSAP.
 *                      Because this session entity is attached to only one
 *                      TSAP, and because X.400 specifies a one-to-one
 *                      mapping between TSAPs and SSAPs, this session entity
 *                      only allows one RTS to register with it. Therefore,
 *                      the SSAPid is strictly unnecessary. If a SSAPid with
 *                      zero length is used, it must be the only one used.
 *            ssu     - a pointer to the SS-user function which handles
 *                      session indication and confirm primitives.
 *
 * OUTPUTS:   returns:  TRUE  if registration was successful,
 *                      FALSE if not.
 *
 * CALLS:     bcopy.
 */

int s_activate(ssap_id,ssu)
register ssap_selector *ssap_id;
register void          (*ssu)();
{
 boolean retval;
 if (nssaps < NSSAPS) {       /* if SS-user may register (only 1) */
    bcopy(ssap_id, &locSSAPid, sizeof(ssap_selector));
    SSuser = ssu;             /* SS-user now active */
    nssaps++;
    retval = TRUE;
 }
 else
    retval = FALSE;

#ifdef DEBUG
  printf("\ns_activate(): Register a SS-user.\n");
  printf("   local SSAPid: ");
  printHEX(ssap_id->addr,ssap_id->len);
  if (retval) printf("   return TRUE\n"); else printf("   return FALSE\n");
#endif

 return(retval);
}
```

```
/*
 * FUNCTION:  s_deactivate
 *
 *            The SS-user calls this function to de-register itself from
 *            the session entity. After this call, the SS-user cannot
 *            initiaite or participate in session connections. If the
 *            SS-user has any active session connections at the time of
 *            the call, the call will fail.
 *
 * INPUTS:    ssap_id - a pointer to the local SSAPid.
 *
 * OUTPUTS:   returns:  TRUE  if de_registration was successful,
 *                      FALSE if not.
 *
 * CALLS:     ENTER, LEAVE, ssap_cmp, SpAc, TDISreq.
 */

int s_deactivate(ssap_id)
register ssap_selector *ssap_id;
{
 boolean retval;
 int     i;
 ENTER();                                  /* disable interrupts */

 retval = !ssap_cmp(&locSSAPid, ssap_id);  /* is SSAPid valid? */

                                           /* if SSAPid valid: */
 for (i = 0; i < NSPMS && retval; i++)     /* search for any active SCs */
     if (SPM[i].STATE != STA01  &&         /* idle, no TC   */
         SPM[i].STATE != STA01B &&         /* await TCONcnf */
         SPM[i].STATE != STA01C   ) {      /* idle, TC con  */
        retval = FALSE;                    /* active SC found */
        break;
     }

                                           /* if no active SCs: */
 for (i = 0; i < NSPMS && retval; i++) {   /* disconnect any active TCs */
     CURNTs = &SPM[i];
     switch (CURNTs->STATE) {
     case STA01C:                          /* idle, TC con  */
          SpAc(32);                        /* stop connect timer */
     case STA01B:                          /* await TCONcnf */
          TDISreq(CURNTs->TCEPid);         /* release TC    */
          TO(STA01);                       /* idle, no TC   */
     }
 }

 if (retval) nssaps--;                     /* SS-user de-registered */

 LEAVE();                                  /* enable interrupts */

#ifdef DEBUG
  printf("\ns_deactivate(): De-register a SS-user.\n");
  printf("   local SSAPid: ");
  printHEX(ssap_id->addr,ssap_id->len);
  if (retval) printf("   return TRUE\n"); else printf("   return FALSE\n");
#endif

 return(retval);
}
```

```
/*
 * FUNCTION:  s_connect
 *
 *            The SS-user calls this function to initiate session connection
 *            establishment. This function attempts to allocate a SPM for
 *            the session connection and, if successful, initiates session
 *            connection establishment.
 *            If reuse of TCs is implemented, it may allocate a free SPM
 *            with a suitable, reusable TC; otherwise, a free SPM with no
 *            TC is allocated, if one is available.
 *
 * INPUTS:    idu - a pointer to the SIDU.
 *
 * OUTPUTS:   returns: the SCEPid if connection establisment initiation
 *                     was successful,
 *                     NULL if not.
 *
 *                     NOTE: The SCEPid is a pointer to the allocated
 *                           SPM structure. However, the SS-user only
 *                           uses this value to identify the SC.
 *
 * CALLS:     ENTER, LEAVE, nsap_cmp, tsap_cmp, ssap_cmp, session.
 *
 *            NOTE: SAP comparison functions return zero if TRUE,
 *                  non-zero if FALSE.
 */

struct Smachine *s_connect(idu)
register struct idu *idu;
{
 struct Smachine *s = NULL;
 int             i;

#ifdef DEBUG
  printf("\ns_connect(): Allocate a SPM for a session connection.\n");
  printf("   local  SSAPid:   ");
  printHEX(idu->loc_ssap->addr,idu->loc_ssap->len);
  printf("   remote NSAPaddr: ");
  printHEX(idu->rem_addr->nsap.addr,idu->rem_addr->nsap.len);
  printf("   remote TSAPid:   ");
  printHEX(idu->rem_addr->tsap.addr,idu->rem_addr->tsap.len);
  printf("   remote SSAPid:   ");
  printHEX(idu->rem_addr->ssap.addr,idu->rem_addr->ssap.len);
#endif

 ENTER();                                /* disable interrupts */

 for (i = 0; i < NSPMS; i++) {           /* search SPMs for re-usable TC */
     CURNTs = &SPM[i];
     if (SPM[i].STATE == STA01C && p(1) &&
         !nsap_cmp(&SPM[i].remSSAPaddr.nsap, &idu->rem_addr->nsap) &&
         !tsap_cmp(&SPM[i].remSSAPaddr.tsap, &idu->rem_addr->tsap) &&
         !ssap_cmp(&SPM[i].remSSAPaddr.ssap, &idu->rem_addr->ssap)   ) {
        s = &SPM[i];                     /* found */

#ifdef DEBUG
  printf("   SPM(%d) with re-usable TC allocated.\n",i);
#endif

        break;
     }
 }

 if (!s)                                 /* if search unsuccessful */
    for (i = 0; i < NSPMS; i++)          /* search SPMs for free TC */
        if (SPM[i].STATE == STA01) {
           s = &SPM[i];                  /* found */
```

```
#ifdef DEBUG
   printf("   SPM(%d) with free TC allocated.\n",i);
#endif

          break;
        }

 if (s)                                  /* if SPM allocated */
    return(session(s,idu) ? s : NULL); /* pass SCONreq onto session() */
 else {

#ifdef DEBUG
  printf("   All SPMs busy - return NULL.\n");
#endif

    LEAVE();                             /* enable interrupts */

    return(NULL);                        /* no SPM allocated - all busy */
 }
}

/*
 * FUNCTION:  session
 *
 *            The SS-user calls this function to pass session request and
 *            response primitives down to the session entity.
 *
 * INPUTS:    s   - the SCEPid (pointer to the SPM). This value was
 *                  returned to the SS-user following a successful call
 *                  to s_connect.
 *            idu - a pointer to the SIDU.
 *
 * OUTPUTS:   returns: TRUE  if the requested event was successful,
 *                     FALSE if not.
 *
 * CALLS:     ENTER, LEAVE, SessionError, balloc, bfree, bcopy, SpAc, p,
 *            Transport request and response event functions,
 *            Session indication and confirm event functions,
 *            SPDU building functions.
 *
 * NOTE: If the RTS call to session fails, idu->buffer is freed if != NULL.
 */

int session(s,idu)
register struct Smachine *s;
register struct idu      *idu;
{
 boolean SCEPidLegal;
 boolean retval;
 int     i;
 CURNTs = s;                    /* current SCEPid */

#ifdef DEBUG
  printf("\nsession(): Session req/rsp primitive received.\n");
  printf("   Input event:  %s\n",Sevent[idu->event]);
#endif

 /*
  * Attempt to validate the SCEPid:
  */

 SCEPidLegal = FALSE;
 for (i = 0; i < NSPMS; i++)
     if (s == &SPM[i]) {
        SCEPidLegal = TRUE;
        break;
     }
```

```
 if (!SCEPidLegal) {          /* if invalid SCEPid */
    SessionError(SERSID);     /* error message */
    return(FALSE);            /* ignore the event */
 }

#ifdef DEBUG
  printf("   SPM:          %d\n",i);
  printf("   From state:   %s\n",Sstate[(int) s->STATE]);
#endif

 /*
  * State transitions:
  */

 ENTER();                     /* disable interrupts */

 switch (s->STATE) {              /*** FROM INITIAL STATE ***/
 case STA01:                      /*** idle, no TC ***/

      switch (idu->event) {
      case SCONREQ:                       /* S-CONNECT.request */

           if ((idu->buffer) || (idu->buffer = balloc(0))) {
              s->TCEPid = TCONreq(&idu->rem_addr->nsap,
                                  &idu->rem_addr->tsap);
              if (s->TCEPid) {

                 /* save remote SSAP address: NSAPaddr, TSAPid, SSAPid */
                 bcopy(idu->rem_addr, &s->remSSAPaddr, sizeof(ssap_address));

                 /* save the SIDU */
                 bcopy(idu, &s->idu, sizeof(struct idu));

                 SpAc(2);                 /* TC initiator is TRUE */
                 TO(STA01B);              /* await TCONcnf */
                 retval = TRUE;           /* SCONreq successful */
                 break;
              }
           }
           retval = FALSE;                /* SCONreq failed */
           break;

      default:                            /* Any other events */
           retval = FALSE;                   /* invalid, so ignore */
           break;
      }
      break;


 case STA01A:                     /*** await ABORT ACCEPT SPDU ***/
      retval = FALSE;                  /* all events invalid, so ignore */
      break;


 case STA01B:                     /*** await T-CONNECT.confirm ***/

      switch (idu->event) {
      case SUABREQ:                       /* S-U-ABORT.request */
           TDISreq(s->TCEPid);               /* release TC */
           TO(STA01);                        /* idle, no TC */
           retval = TRUE;                    /* SUABreq successful */
           break;

      default:                            /* Any other events */
           retval = FALSE;                   /* invalid, so ignore */
           break;
      }
      break;
```

```
case STA01C:                    /*** idle, TC con ***/

     switch (idu->event) {
     case SCONREQ:                      /* S-CONNECT.request */
          if (p(1)) {                   /* if TC initiator */
             if ((idu->buffer) || (idu->buffer = balloc(0))) {
                s->fus = idu->fu;

                BuildCN(idu->buffer,          /* TSDU with SS-user data */
                        &idu->scid,           /* Session Connection id */
                        PROTOCOL,             /* protocol options */
                        MAXTSDULEN,           /* max TSDU len, I to R */
                        MAXTSDULEN,           /* max TSDU len, R to I */
                        VERSION,              /* version number */
                        idu->sn,              /* initial serial number */
                        idu->token,           /* initial token assignments */
                        idu->fu,              /* Srequirements */
                        &locSSAPid,           /* calling SSAPid (loc) */
                        &s->remSSAPaddr.ssap); /* called  SSAPid (rem) */

                TDTreq(s->TCEPid,             /* send CN */
                       idu->buffer);
                SDTcnf(idu->buffer);          /* free TSDU buffer */
                SpAc(32);                     /* stop connect timer */
                TO(STA02A);                   /* await ACCEPT SPDU */
                retval = TRUE;                /* SCONreq successful */
                break;
             }
             retval = FALSE;                  /* SCONreq failed */
             break;
          }

                                        /* if not TC initiator */
          retval = FALSE;               /* SCONreq failed */
          break;

     default:                           /* Any other events */
          retval = FALSE;               /* invalid, so ignore */
          break;
     }
     break;


case STA02A:                    /*** await ACCEPT SPDU ***/

     switch (idu->event) {
     case SUABREQ:                      /* S-U-ABORT.request */
          if (!p(2)) {                  /* if TC may not be reused */
             if (UserAbort(s,idu,FALSE)) {  /* issue ABORTnr SPDU */
                TO(STA16);              /* await TDISind */
                retval = TRUE;          /* SUABreq successful */
                break;
             }
             retval = FALSE;            /* SUABreq failed */
             break;
          }

          if (p(2)) {                   /* if TC may be reused */
             if (UserAbort(s,idu,TRUE)) {   /* issue ABORTr SPDU */
                TO(STA01A);             /* await ABORT ACCEPT SPDU */
                retval = TRUE;          /* SUABreq successful */
                break;
             }
             retval = FALSE;            /* SUABreq failed */
             break;
          }
```

```
    default:                          /* Any other events */
        retval = FALSE;                  /* invalid, so ignore */
        break;
    }
    break;


case STA08:                       /*** await S-CONNECT.response ***/

    switch (idu->event) {
    case SCONACC:                     /* S-CONNECT.response (accept) */
        if ((idu->buffer) || (idu->buffer = balloc(0))) {
           uint8 token,               /* token indentifier */
                 mask,                /* 2-bit token assignment mask */
                 AccChooses,          /* ACCEPTOR CHOOSES token assigment */
                 AccSide,             /* ACCEPTOR SIDE token assignment */
                 TokenSetItem;        /* Token Setting Item for AC */

           /*
            * FUs selected for this SC are those proposed by both SSUs
            */
           s->fus &= idu->fu;

           /*
            * Determine the available tokens on this SC:
            */
           s->AvTokens = 0;
           for (token = DKT; token & TDM; token <<= 2)
               if (AV(token))
                  s->AvTokens |= token;

           /*
            * Determine Token Setting Item for AC and
            * tokens assigned to this SPM:
            */
           s->AsTokens   = 0;
           TokenSetItem  = 0;
           token         = DKT;
           mask          = 0x03;     /* isolate 2 low-order bits */
           AccChooses    = 0x02;
           AccSide       = 0x01;
           while (token & TDM) {
                 if (s->TEMPtoken & mask == AccChooses)
                    TokenSetItem |= idu->token & mask;
                 else
                    TokenSetItem |= s->TEMPtoken & mask;

                 if (AV(token) && TokenSetItem & mask == AccSide)
                    s->AsTokens |= token;
                 token      <<= 2;     /* next token */
                 mask       <<= 2;     /* next token assignment mask */
                 AccChooses <<= 2;     /* next Acceptor Chooses assignment */
                 AccSide    <<= 2;     /* next Acceptor Side assignment */
           }

           BuildAC(idu->buffer,          /* TSDU with SS-user data */
                   &idu->scid,           /* SC identifier */
                   PROTOCOL,             /* protocol options */
                   MAXTSDULEN,           /* max TSDU len, I to R */
                   MAXTSDULEN,           /* max TSDU len, R to I */
                   VERSION,              /* version number */
                   idu->sn,              /* initial serial number */
                   TokenSetItem,         /* token setting item */
                   0,                    /* token item */
                   idu->fu,              /* session requirements */
                   &s->remSSAPaddr.ssap, /* calling SSAP id (rem) */
                   &locSSAPid);          /* called  SSAP id (loc) */
```

```
            TDTreq(s->TCEPid,              /* send AC */
                   idu->buffer);
            bfree(idu->buffer);            /* free TSDU buffer */
            SpAc(5, NULL, idu->sn, s->Texp, s->fus);
            SpAc(11, s->AsTokens);
            TO(STA713);                    /* data transfer */
            retval = TRUE;                 /* SCONrsp+ successful */
            break;
         }
         retval = FALSE;                   /* SCONrsp+ failed */
         break;

   case SCONREJ:                    /* S-CONNECT.response (reject) */

         if (!p(2)) {                    /* if TC may not be reused */
            if ((idu->buffer) || (idu->buffer = balloc(0))) {
               struct TCdis TCdis;
               TCdis.ABreason = NO_ABORT; /* no Abort in progress */
               TCdis.TCkept   = FALSE;    /* for RFnr */

               BuildRF(idu->buffer,       /* TSDU with SS-user Data */
                       &idu->scid,        /* SC identifier */
                       &TCdis,            /* transport disconnect */
                       idu->fu,           /* session requirements */
                       VERSION,           /* version number */
                       idu->reason);      /* reason */

               TDTreq(s->TCEPid,          /* send RF */
                      idu->buffer);
               SDTcnf(idu->buffer);       /* free TSDU buffer */
               SpAc(4);                   /* start abort timer */
               TO(STA16);                 /* await TDISind */
               retval = TRUE;             /* SCONrsp- successful */
               break;
            }
            retval = FALSE;               /* SCONrsp- failed */
            break;
         }

         if (p(2)) {                     /* if TC may be reused */
            if ((idu->buffer) || (idu->buffer = balloc(0))) {
               struct TCdis TCdis;
               TCdis.ABreason = NO_ABORT; /* no Abort in progress */
               TCdis.TCkept   = TRUE;     /* for RFr */

               BuildRF(idu->buffer,       /* TSDU with SS-user Data */
                       &idu->scid,        /* Session Connection id */
                       &TCdis,            /* transport disconnect */
                       idu->fu,           /* session requirements */
                       VERSION,           /* version number */
                       idu->reason);      /* reason */

               TDTreq(s->TCEPid,          /* send RF */
                      idu->buffer);
               SDTcnf(idu->buffer);       /* free TSDU buffer */
               SpAc(33);                  /* start connect timer */
               TO(STA01C);                /* idle, TC con */
               retval = TRUE;             /* SCONrsp- successful */
               break;
            }
            retval = FALSE;               /* SCONrsp- failed */
            break;
         }
```

```
        case SUABREQ:                    /* S-U-ABORT.request */
             if (!p(2)) {                  /* if TC may not be reused */
                if (UserAbort(s,idu,FALSE)) { /* issue ABORTnr SPDU */
                   TO(STA16);              /* await TDISind */
                   retval = TRUE;          /* SUABreq successful */
                   break;
                }
                retval = FALSE;            /* SUABreq failed */
                break;
             }

             if (p(2)) {                  /* if TC may be reused */
                if (UserAbort(s,idu,TRUE)) {  /* issue ABORTr SPDU */
                   TO(STA01A);             /* await AA */
                   retval = TRUE;          /* SUABreq successful */
                   break;
                }
                retval = FALSE;            /* SUABreq failed */
                break;
             }

        default:                         /* Any other events */
             retval = FALSE;               /* invalid, so ignore */
             break;
        }
        break;


   case STA16:                       /*** await T-DISCONNECT.indication ***/
        retval = FALSE;                   /* all events invalid, so ignore */
        break;


   case STA713:                      /*** data transfer */

        switch (idu->event) {

        /*
         * For testing purposes, ignore the data transfer primitives SACTSreq,
         * SDTreq and SSYNmreq, and abort the TC when SACTEreq is received.
         */

        case SACTSREQ:                    /* S-ACTIVITY-START.request */
        case SDTREQ:                      /* S-DATA.request */
        case SSYNREQ:                     /* S-SYNC-MINOR.request */
             if (idu->buffer)
                bfree(idu->buffer);
             TO(SAME);                       /* data transfer */
             retval = TRUE;
             break;

        case SACTEREQ:                    /* S-ACTIVITY-END.request */
             if (idu->buffer)
                bfree(idu->buffer);
             TDISind((pointer) 2,0,NULL);   /* abort the TC at both ends */
             TDISreq(s->TCEPid);
             TO(SAME);                       /* data transfer */
             retval = TRUE;
             break;

        default:                          /* all other S events */
             retval = FALSE;                 /* transitions not implemented */
             break;
        }
        break;
   }

   LEAVE();                       /* enable interrupts */
```

```
#ifdef DEBUG
  printf("   To state:     %s\n",Sstate[(int) s->STATE]);
  if (retval) printf("   return TRUE\n"); else printf("   return FALSE\n");
#endif

 return(retval);
}

/*
 * FUNCTION:  do_session_queue
 *
 *            The SS-user calls this function to allow the session
 *            entity to processes the queue of transport indication
 *            and confirm primitives. This function returns once all the
 *            events in the queue have been removed and processed and
 *            the queue is empty.
 *
 * INPUTS:    none.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     ENTER, LEAVE, SessionError, QRemove, bfree, os_free, bcopy,
 *            SpAc, p,
 *            Transport request and response event functions,
 *            Session indication and confirm event functions,
 *            SPDU building and stripping functions.
 */

void do_session_queue()
{
 TQelement       *TQelem;  /* Transport event queue element */
 struct Smachine *s;       /* SCEPid */
 int             i;

 /* Get transport events from inter-process message queue: */
 do_transport_queue();

 while ((TQelem = TQhead.last) != (TQelement*) &TQhead) {

      ENTER();                    /* disable interrupts */

#ifdef DEBUG
  printf("\ndo_session_queue(): Transport ind/cnf primitive received.\n");
  printf("   Input event:  %s\n",Tevent[TQelem->event]);
  if (TQelem->event == TDTIND) printSPDU(TQelem->tsdu);
#endif

       /*
        * If TCONind received, try to allocate a SPM for the TC:
        */

       if (TQelem->event == TCONIND) {
          boolean SPMalloc = FALSE;

#ifdef DEBUG
  printf("         remote NSAPaddr: ");
  printHEX(TQelem->clgNSAPaddr.addr,TQelem->clgNSAPaddr.len);
  printf("         remote TSAPid:   ");
  printHEX(TQelem->clgTSAPid.addr,TQelem->clgTSAPid.len);
#endif

          for (i = 0; i < NSPMS; i++)        /* search for free SPM */
              if (SPM[i].STATE == STA01) {   /* if free SPM found */
                 SPM[i].TCEPid = TQelem->TCEPid;
                 SPMalloc = TRUE;            /* SPM is allocated */
                 break;
              }
```

```
            if (!SPMalloc) {              /* no SPM allocated - all busy */
               QRemove(TQelem);           /* de-queue the event */
               os_free(TQelem);           /* free element memory */
               TDISreq(TQelem->TCEPid);   /* reject the TC */
               LEAVE();                   /* enable interrupts */
               continue;                  /* ignore the event */
            }
         }

         /*
          * Attempt to validate the incoming event's TCEPid by searching
          * the SPMs for its matching SCEPid:
          */

         s = NULL;
         for (i = 0; i < NSPMS; i++)
             if (SPM[i].TCEPid == TQelem->TCEPid) {
                s = &SPM[i];              /* SPM found */
                break;
             }

         if (!s) {                        /* if TCEPid invalid */
            SessionError(SERTID);         /* error message */
            QRemove(TQelem);              /* de-queue the event */
            if (TQelem->tsdu)
               bfree(TQelem->tsdu);       /* free possible TSDU buffer */
            os_free(TQelem);              /* free element memory */
            LEAVE();                      /* enable interrupts */
            continue;                     /* ignore the event */
         }

         /*
          * State transitions:
          */

         CURNTs = s;                      /* current SCEPid */

#ifdef DEBUG
  printf("   SPM:          %d\n",i);
  printf("   From state:   %s\n",Sstate[(int) s->STATE]);
#endif

         switch (s->STATE) {      /*** FROM INITIAL STATE ***/
         case STA01:              /*** idle, no TC ***/
             switch (TQelem->event) {

             case TDTIND:                 /* T-DATA.indication: any SPDU event */
                  bfree(TQelem->tsdu);    /* invalid, so ignore */
                  break;

             case TCONIND:                /* T-CONNECT.indication */

                  /* save remote TSAP address: NSAPaddr, TSAPid */
                  bcopy(&TQelem->clgNSAPaddr, &s->remSSAPaddr.nsap,
                        sizeof(nsap_address));
                  bcopy(&TQelem->clgTSAPid, &s->remSSAPaddr.tsap,
                        sizeof(tsap_selector));

                  TCONrsp(TQelem->TCEPid); /* accept the TC */
                  SpAc(1);                 /* TC initiator is FALSE */
                  SpAc(33);                /* start connect timer */
                  TO(STA01C);              /* idle, TC con */
                  break;

             case TCONCNF:                /* T-CONNECT.confirm */
             case TDISIND:                /* T-DISCONNECT.indication */
                  break;                  /* invalid, so ignore */
             }
             break;
```

```
case STA01A:            /*** await ABORT ACCEPT SPDU ***/
     switch (TQelem->event) {
     case TDTIND:            /* T-DATA.indication: SPDU event */

          if (idSPDU(TQelem->tsdu,RF) ||
              idSPDU(TQelem->tsdu,AC)   ) {
             TO(SAME);
             bfree(TQelem->tsdu);
             break;
          }

          if (idSPDU(TQelem->tsdu,CN) ||
              idSPDU(TQelem->tsdu,ABnr) ) {
             TDISreq(TQelem->TCEPid);
             SpAc(3);              /* stop abort timer */
             TO(STA01);            /* idle, no TC */
             bfree(TQelem->tsdu);
             break;
          }

          if (idSPDU(TQelem->tsdu,AA) ||
              idSPDU(TQelem->tsdu,ABr)  ) {
             SpAc(3);              /* stop abort timer */
             SpAc(33);             /* start connect timer */
             TO(STA01C);           /* idle, TC con */
             bfree(TQelem->tsdu);
             break;
          }
                              /* Any other SPDU */
          bfree(TQelem->tsdu);     /* invalid, so ignore */
          break;

     case TCONIND:           /* T-CONNECT.indication */
     case TCONCNF:           /* T-CONNECT.confirm */
          break;                /* invalid, so ignore */

     case TDISIND:           /* T-DISCONNECT.indication */
          SpAc(3);              /* stop abort timer */
          TO(STA01);            /* idle, no TC */
          break;
     }
     break;

case STA01B:            /*** await TCONcnf ***/
     switch (TQelem->event) {

     case TDTIND:            /* T-DATA.indication: any SPDU event */
          bfree(TQelem->tsdu);  /* invalid, so ignore */
          break;

     case TCONIND:           /* T-CONNECT.indication */
          break;                /* invalid, so ignore */

     case TCONCNF:           /* T-CONNECT.confirm */
          s->Texp = TQelem->texp;        /* selected TEXP option */
          s->fus = s->idu.fu;            /* proposed Srequirements */

          BuildCN(s->idu.buffer,         /* TSDU with SS-user data */
                  &s->idu.scid,          /* Session Connection id */
                  PROTOCOL,              /* protocol options */
                  MAXTSDULEN,            /* max TSDU len, I to R */
                  MAXTSDULEN,            /* max TSDU len, R to I */
                  VERSION,               /* version number */
                  s->idu.sn,             /* initial serial number */
                  s->idu.token,          /* initial token assignments */
                  s->idu.fu,             /* Srequirements */
                  &locSSAPid,            /* calling SSAPid (loc) */
                  &s->remSSAPaddr.ssap); /* called  SSAPid (rem) */
```

```
            TDTreq(s->TCEPid,                /* send CN */
                   s->idu.buffer);
            SDTcnf(s->idu.buffer);           /* free the buffer */
            TO(STA02A);                      /* await AC */
            break;

        case TDISIND:           /* T-DISCONNECT.indication */
            SPABind(T_DISCONNECT);    /* abort the SC */
            TO(STA01);                /* idle, no TC */
            break;
        }
        break;

    case STA01C:            /*** idle, TC con ***/
        switch (TQelem->event) {
        case TDTIND:            /* T-DATA.indication: SPDU event */

            if (idSPDU(TQelem->tsdu,RF)           ||
                idSPDU(TQelem->tsdu,CN)  && p(1)  ||
                idSPDU(TQelem->tsdu,AC)           ||
                idSPDU(TQelem->tsdu,AA)           ||
                idSPDU(TQelem->tsdu,ABnr)         ||
                idSPDU(TQelem->tsdu,ABr) && !p(2)   ) {
               TDISreq(TQelem->TCEPid);
               SpAc(32);                /* stop connect timer */
               TO(STA01);               /* idle, no TC */
               bfree(TQelem->tsdu);
               break;
            }

            if (idSPDU(TQelem->tsdu,CN) && !p(1)) {
               struct idu    idu;
               uint8         protocol;
               uint16        max0TSDUlen,max1TSDUlen;
               ssap_selector cldSSAPid;

               idu.fu = 0;                    /* int is 32 bits on 80386 */
               StripCN(TQelem->tsdu,          /* TSDU buffer */
                       &idu.scid,             /* SC identifier */
                       &protocol,             /* protocol options */
                       &max0TSDUlen,          /* max TSDU len I to R */
                       &max1TSDUlen,          /* max TSDU len R to I */
                       &idu.version,          /* version number */
                       &idu.sn,               /* initial serial number */
                       &idu.token,            /* token setting item */
                       &idu.fu,               /* Srequirements */
                       &s->remSSAPaddr.ssap, /* calling SSAP id (rem) */
                       &cldSSAPid,            /* called  SSAP id (loc) */
                       &idu.buffer);          /* SS-user data */
```

```
/*** FOR TESTING ONLY ***/
     printf("\n   SPDU parameters:\n");
     printf("   scid.clg_ref = ");
     printHEX(idu.scid.clg_ref.data,idu.scid.clg_ref.len);
     printf("   scid.com_ref = ");
     printHEX(idu.scid.com_ref.data,idu.scid.com_ref.len);
     printf("   scid.add_ref = ");
     printHEX(idu.scid.add_ref.data,idu.scid.add_ref.len);
     printf("   protocol     = %d\n",protocol);
     printf("   max0TSDUlen  = %d\n",max0TSDUlen);
     printf("   max1TSDUlen  = %d\n",max1TSDUlen);
     printf("   version      = %d\n",idu.version);
     printf("   token        = %d\n",idu.token);
     printf("   initial sn   = %d\n",idu.sn);
     printf("   fus          = %d\n",idu.fu);
     printf("   clgSSAPid    = ");
     printHEX(s->remSSAPaddr.ssap.addr,s->remSSAPaddr.ssap.len);
     printf("   cldSSAPid    = ");
     printHEX(cldSSAPid.addr,cldSSAPid.len);
     printf("   SS-user data:\n");
     printDEC(idu.buffer->addr,idu.buffer->length);

     idu.fu = 0; /* Insert this statement to cause the called
                  * RTS to reject the SC establishment attempt.
                  */
/*** FOR TESTING ONLY ***/

     /* save the initiator's protocol options: */
     s->protocol = protocol;

     /*
      * For each direction of transfer, the lesser value
      * for max TSDU len, proposed by each SPM, is used:
      */
     s->max0TSDUlen = min(max0TSDUlen,MAXTSDULEN);
     s->max1TSDUlen = min(max1TSDUlen,MAXTSDULEN);

     /*
      * The highest common version number
      * proposed by the SPMs is used:
      */
     s->version = min(idu.version,VERSION);

     /* proposed functional units */
     s->fus = idu.fu;

     /*
      * Temporarily save the proposed initial token
      * assignments for later use if SCONrsp+ is received:
      */
     s->TEMPtoken = idu.token;

     SCONind(&idu);              /* issue SCONind */

     if (idu.buffer == NULL)
        bfree(TQelem->tsdu);     /* no SS-user data */

     SpAc(32);                   /* stop connect timer */
     TO(STA08);                  /* await SCONrsp */
     break;
  }
```

```
            if (idSPDU(TQelem->tsdu,ABr) && p(2)) {
               bclear(TQelem->tsdu);      /* reuse buffer */
               BuildAA(TQelem->tsdu);
               TDTreq(s->TCEPid,          /* send AA */
                     TQelem->tsdu);
               bfree(TQelem->tsdu);
               SpAc(33);                  /* start connect timer */
               TO(STA01C);                /* idle, TC con */
               break;
            }

                                    /* PROVIDED OTHERWISE */
            bfree(TQelem->tsdu);       /* invalid, so ignore */
            break;

         case TCONIND:              /* T-CONNECT.indication */
         case TCONCNF:              /* T-CONNECT.confirm */
            break;                     /* invalid, so ignore */

         case TDISIND:              /* T-DISCONNECT.indication */
            TO(STA01);                 /* idle, no TC */
            break;
         }
         break;

      case STA02A:              /*** await AC ***/
         switch(TQelem->event) {
         case TDTIND:               /* T-DATA.indication: SPDU event */

            if (idSPDU(TQelem->tsdu,RFnr) ||
                idSPDU(TQelem->tsdu,RFr)  && !p(2)) {
               struct idu   idu;
               struct TCdis TCdis;

               idu.fu = 0;              /* int is 32 bits on 80386 */
               StripRF(TQelem->tsdu,    /* TSDU buffer */
                       &idu.scid,       /* SC identifier */
                       &TCdis,          /* transport disconnect */
                       &idu.fu,         /* Srequirements */
                       &idu.version,    /* version number */
                       &idu.reason,     /* reason code */
                       &idu.buffer);    /* SS-user data */

               SCONcnfrej(&idu);        /* issue SCONcnf- */
               TDISreq(s->TCEPid);
               if (idu.buffer == NULL)
                  bfree(TQelem->tsdu);  /* no SS-user data */
               TO(STA01);               /* idle, no TC */
               break;
            }

            if (idSPDU(TQelem->tsdu,RFr) && p(2)) {
               struct idu   idu;
               struct TCdis TCdis;

               idu.fu = 0;              /* int is 32 bits on 80386 */
               StripRF(TQelem->tsdu,    /* TSDU buffer */
                       &idu.scid,       /* SC identifier */
                       &TCdis,          /* transport disconnect */
                       &idu.fu,         /* Srequirements */
                       &idu.version,    /* version number */
                       &idu.reason,     /* reason code */
                       &idu.buffer);    /* SS-user data */
```

```
        /*
         * The highest common version number
         * proposed by the SPMs is used:
         */
        s->version = min(idu.version,VERSION);

        /*
         * The functional units selected for use on
         * this SC are those proposed by both SS-users:
         */
        s->fus &= idu.fu;

        /*
         * The available tokens on this SC:
         */
        s->AvTokens = 0;
        for (token = DKT; token & TDM; token <<= 2)
            if (AV(token))
               s->AvTokens |= token;

        /*
         * The tokens assigned to the requestor:
         */
        s->AsTokens = 0;
        for (token = DKT, mask = 0x03; token & TDM;
             token <<= 2, mask <<= 2)
            if (AV(token) && !(idu.token & mask))
               s->AsTokens |= token;

        SCONcnfacc(&idu);          /* issue SCONcnf+ */
        SpAc(5, NULL, idu.sn, s->Texp, s->fus);
        SpAc(11, s->AsTokens);
        if (idu.buffer == NULL)
           bfree(TQelem->tsdu);    /* no SS-user data */
        TO(STA713);                /* data transfer */
        break;
    }

    if (idSPDU(TQelem->tsdu,ABnr) ||
        idSPDU(TQelem->tsdu,ABr)  && !p(2)) {
        struct buf    *buffer;
        struct TCdis  TCdis;
        struct Bytes9 ReflectParams;

        StripAB(TQelem->tsdu,       /* TSDU buffer */
                &TCdis,             /* TC disconnect */
                &ReflectParams,     /* reflect parameter values */
                &buffer);           /* SS-user data */
        /*
         * generate the event SUABind if the TCdisconnect PV
         * field in AB has the value USER_ABORT.
         * Otherwise, generate the event SPABind.
         */
        if (TCdis.ABreason == USER_ABORT)
           SUABind(buffer);         /* issue SUABind */
        else
           SPABind(TCdis.ABreason); /* issue SPABind */

        if (buffer == NULL)
           bfree(TQelem->tsdu);
        TDISreq(TQelem->TCEPid);
        TO(STA01);                  /* idle, no TC */
        break;
    }
```

```
            if (idSPDU(TQelem->tsdu,ABr) && p(2)) {
               struct buf    *tsdu;        /* for AA SPDU */
               struct buf    *buffer;      /* for SS-user data */
               struct TCdis  TCdis;
               struct Bytes9 ReflectParams;

               StripAB(TQelem->tsdu,       /* TSDU buffer */
                       &TCdis,             /* TC disconnect */
                       &ReflectParams,     /* reflect parameter values */
                       &buffer);           /* SS-user data */
               /*
                * generate the event SUABind if the TCdisconnect PV
                * field in AB has the value USER_ABORT.
                * Otherwise, generate the event SPABind.
                */
               if (TCdis.ABreason == USER_ABORT)
                  SUABind(buffer);         /* issue SUABind */
               else
                  SPABind(TCdis.ABreason); /* issue SPABind */

               if (buffer == NULL)
                  bfree(TQelem->tsdu);

               tsdu = get_buf(0);
               BuildAA(tsdu);
               TDTreq(s->TCEPid,           /* send AA */
                      tsdu);
               bfree(tsdu);
               SpAc(33);                   /* start connect timer */
               TO(STA01C);                 /* idle, TC con */
               break;
            }
                                  /* PROVIDED OTHERWISE */
            bfree(TQelem->tsdu);     /* invalid, so ignore */
            break;

         case TCONIND:            /* T-CONNECT.indication */
         case TCONCNF:            /* T-CONNECT.confirm */
            break;                /* invalid, so ignore */

         case TDISIND:            /* T-DISCONNECT.indication */
            SPABind(T_DISCONNECT); /* abort the SC */
            TO(STA01);             /* idle, no TC */
            break;
         }
         break;

      case STA08:              /*** await SCONrsp ***/
         switch (TQelem->event) {
         case TDTIND:          /* T-DATA.indication: SPDU event */
            if (idSPDU(TQelem->tsdu,ABnr) ||
                idSPDU(TQelem->tsdu,ABr)  && !p(2)) {
               struct buf    *buffer;      /* for SS-user data */
               struct TCdis  TCdis;
               struct Bytes9 ReflectParams;

               StripAB(TQelem->tsdu,       /* TSDU buffer */
                       &TCdis,             /* TC disconnect */
                       &ReflectParams,     /* reflect parameter values */
                       &buffer);           /* SS-user data */
```

```
            /*
             * generate the event SUABind if the TCdisconnect PV
             * field in AB has the value USER_ABORT.
             * Otherwise, generate the event SPABind.
             */
            if (TCdis.ABreason == USER_ABORT)
               SUABind(buffer);          /* issue SUABind */
            else
               SPABind(TCdis.ABreason); /* issue SPABind */

            if (buffer == NULL)
               bfree(TQelem->tsdu);
            TDISreq(TQelem->TCEPid);
            TO(STA01);                   /* idle, no TC */
            break;
         }

         if (idSPDU(TQelem->tsdu,ABr) && p(2)) {
            struct buf    *tsdu;         /* for AA SPDU */
            struct buf    *buffer;       /* for SS-user data */
            struct TCdis  TCdis;
            struct Bytes9 ReflectParams;
            StripAB(TQelem->tsdu,        /* TSDU buffer */
                    &TCdis,              /* TC disconnect */
                    &ReflectParams,      /* reflect parameter values */
                    &buffer);            /* SS-user data */
            /*
             * generate the event SUABind if the TCdisconnect PV
             * field in AB has the value USER_ABORT.
             * Otherwise, generate the event SPABind.
             */
            if (TCdis.ABreason == USER_ABORT)
               SUABind(buffer);          /* issue SUABind */
            else
               SPABind(TCdis.ABreason); /* issue SPABind */

            if (buffer == NULL)
               bfree(TQelem->tsdu);

            tsdu = get_buf(0);
            BuildAA(tsdu);
            TDTreq(s->TCEPid,            /* send AA */
                 tsdu);
            bfree(tsdu);
            SpAc(33);                    /* start connect timer */
            TO(STA01C);                  /* idle, TC con */
            break;
         }
                              /* PROVIDED OTHERWISE */
         bfree(TQelem->tsdu);     /* invalid, so ignore */
         break;

      case TCONIND:           /* T-CONNECT.indication */
      case TCONCNF:           /* T-CONNECT.confirm */
         break;

      case TDISIND:           /* T-DISCONNECT.indication */
         SPABind(T_DISCONNECT); /* abort SC */
         TO(STA01);              /* idle, no TC */
         break;
      }
      break;
```

```
        case STA16:             /*** await TDISind ***/
             switch (TQelem->event) {
             case TDTIND:            /* T-DATA.indication: SPDU event */

                  if (idSPDU(TQelem->tsdu,RF) ||
                      idSPDU(TQelem->tsdu,AC)   ) {
                     TO(SAME);
                     bfree(TQelem->tsdu);
                     break;
                  }

                  if (idSPDU(TQelem->tsdu,CN) ||
                      idSPDU(TQelem->tsdu,AA) ||
                      idSPDU(TQelem->tsdu,AB)   ) {
                     TDISreq(TQelem->TCEPid);
                     SpAc(3);              /* stop abort timer */
                     TO(STA01);            /* idle, no TC */
                     bfree(TQelem->tsdu);
                     break;
                  }
                                       /* PROVIDED OTHERWISE */
                  bfree(TQelem->tsdu);    /* invalid, so ignore */
                  break;

             case TCONIND:           /* T-CONNECT.indication */
             case TCONCNF:           /* T-CONNECT.confirm */
                  break;                /* invalid, so ignore */

             case TDISIND:           /* T-DISCONNECT.indication */
                  SpAc(3);              /* stop abort timer*/
                  TO(STA01);            /* idle, no TC */
                  break;
             }
             break;

        case STA713:            /*** data transfer ***/
             switch (TQelem->event) {
             case TDTIND:            /* WHEN T-DATA.indication */
             case TCONIND:           /* WHEN T-CONNECT.indication */
             case TCONCNF:           /* WHEN T-CONNECT.confirm */
                  break;

             case TDISIND:           /* T-DISCONNECT.indication */
                  SPABind(T_DISCONNECT); /* abort SC */
                  TO(STA01);             /* idle, no TC */
                  break;
             }
             break;
        }                       /* switch (s->STATE) */

        LEAVE();                /* enable interrupts */

        QRemove(TQelem);        /* de-queue the event */
        os_free(TQelem);        /* free element memory */

#ifdef DEBUG
  printf("   To state:     %s\n",Sstate[(int) s->STATE]);
#endif

  }                             /* while (): process next event */
}
```

```
/*
 * FUNCTIONs:  TCONind TCONcnf TDISind TDTind
 *
 *             These functions are the transport indication/confirm
 *             primitive handlers. They each place their event on the
 *             Transport Layer Event Queue for later processing by a
 *             SS-user call to so_session_queue().
 *
 * INPUTS:     possible inputs are:
 *             TCEPid      - the TCEPid.
 *             clgNSAPaddr - calling NSAP address.
 *             clgTSAPid   - calling TSAP identifier.
 *             cldNSAPaddr - called  NSAP address.
 *             cldTSAPid   - called  TSAP identifier.
 *             qots        - QOTS values.
 *             texp        - transport expedited data option.
 *             eotsdu      - End Of TSDU flag.
 *             TSUdata     - if non-null, a pointer to TS-user data buffer.
 *
 * OUTPUTS:    returns: TRUE  if successful,
 *                      FALSE if not.
 *
 * CALLS:      get_mem, get_buf, QInit, QInsert, bcopy.
 */

/*
 * T-CONNECT.indication handler
 */

int TCONind(TCEPid,clgNSAPaddr,clgTSAPid,cldTSAPid,qots,texp,TSUdata)
pointer       TCEPid;
nsap_address  *clgNSAPaddr;
tsap_selector *clgTSAPid,
              *cldTSAPid;
qos_type      *qots;
boolean       texp;
uint8         *TSUdata;
{
 TQelement *TQelem = (TQelement*) get_mem(sizeof(TQelement));
 QInit(TQelem);
 TQelem->event   = TCONIND;
 TQelem->TCEPid  = TCEPid;
 TQelem->texp    = texp;
 TQelem->TSUdata = TSUdata;
 TQelem->tsdu    = NULL;
 bcopy(clgNSAPaddr, &TQelem->clgNSAPaddr, sizeof(nsap_address));
 bcopy(clgTSAPid,   &TQelem->clgTSAPid,   sizeof(tsap_selector));
 bcopy(cldTSAPid,   &TQelem->cldTSAPid,   sizeof(tsap_selector));
 bcopy(qots,        &TQelem->qots,        sizeof(qos_type));
 QInsert(&TQhead,TQelem);
 return(TRUE);
}
```

```
/*
 * T-CONNECT.confirm handler
 */

int TCONcnf(TCEPid,cldNSAPaddr,cldTSAPid,qots,texp,TSUdata)
pointer       TCEPid;
nsap_address  *cldNSAPaddr;
tsap_selector *cldTSAPid;
qos_type      *qots;
boolean       texp;
uint8         *TSUdata;
{
 TQelement *TQelem = (TQelement*) get_mem(sizeof(TQelement));
 QInit(TQelem);
 TQelem->event   = TCONCNF;
 TQelem->TCEPid  = TCEPid;
 TQelem->texp    = texp;
 TQelem->TSUdata = TSUdata;
 TQelem->tsdu    = NULL;
 bcopy(cldNSAPaddr, &TQelem->cldNSAPaddr, sizeof(nsap_address));
 bcopy(cldTSAPid,   &TQelem->cldTSAPid,   sizeof(tsap_selector));
 bcopy(qots,        &TQelem->qots,        sizeof(qos_type));
 QInsert(&TQhead,TQelem);
 return(TRUE);
}

/*
 * T-DISCONNECT.indication handler
 */

int TDISind(TCEPid,reason,TSUdata)
pointer TCEPid;
uint8   reason;
uint8   *TSUdata;
{
 TQelement *TQelem = (TQelement*) get_mem(sizeof(TQelement));
 QInit(TQelem);
 TQelem->event   = TDISIND;
 TQelem->TCEPid  = TCEPid;
 TQelem->reason  = reason;
 TQelem->TSUdata = TSUdata;
 TQelem->tsdu    = NULL;
 QInsert(&TQhead,TQelem);
 return(TRUE);
}

/*
 * T-DATA.indication handler
 */

int TDTind(TCEPid,tsdu,eotsdu)
pointer    TCEPid;
struct buf *tsdu;
boolean    eotsdu;
{
 TQelement *TQelem = (TQelement*) get_mem(sizeof(TQelement));
 TQelem->tsdu = get_buf(tsdu->length);
 QInit(TQelem);
 TQelem->event  = TDTIND;
 TQelem->TCEPid = TCEPid;
 TQelem->eotsdu = eotsdu;
 bcopy(tsdu->addr, TQelem->tsdu->addr, tsdu->length);
 QInsert(&TQhead,TQelem);
 return(TRUE);
}
```

E.6 Debug source file listing: debug.c

```
/*
 * SESSION LAYER DEBUG SOURCE FILE: debug.c
 * ED van der Westhuizen   September 1989
 *
 * This file contains static data objects and functions used by
 * session debug statements.
 */

/*
 * SPM state names
 */

static char *Sstate[] = {
                "STA01 - idle, no TC",                      /* 0 */
                "STA01A - await ABORT ACCEPT SPDU",         /* 1 */
                "STA01B - await T-CONNECT.confirm",         /* 2 */
                "STA01C - idle, TC connected",              /* 3 */
                "STA02A - await ACCEPT SPDU",               /* 4 */
                "STA08 - await S-CONNECT.response",         /* 5 */
                "STA16 - await T-DISCONNECT.indication",    /* 6 */
                "STA713 - data transfer"                    /* 7 */
            };

/*
 * Session primitive names
 */

static char *Sevent[] = {
                "illegal",                                  /*  0 */
                "S-ACTIVITY-DISCARD.request",               /*  1 */
                "S-ACTIVITY-DISCARD.response",
                "S-ACTIVITY-END.request",
                "S-ACTIVITY-END.response",
                "S-ACTIVITY-INTERRUPT.request",
                "S-ACTIVITY-INTERRUPT.response",
                "S-ACTIVITY-RESUME.request",
                "S-ACTIVITY-START.request",
                "S-CAPABILITY-DATA.request",
                "S-CAPABILITY-DATA.response",
                "S-CONTROL-GIVE.request",
                "S-CONNECT.request",
                "S-CONNECT.response (accept)",
                "S-CONNECT.response (reject)",
                "S-DATA.request",
                "S-EXPEDITED-DATA.request",
                "S-GIVE-TOKENS.request",
                "S-PLEASE-TOKENS.request",
                "S-RELEASE.request",
                "S-RELEASE.response (accept)",
                "S-RELEASE.response (reject)",
                "S-RESYNCHRONIZE.request (abandon)",
                "S-RESYNCHRONIZE.request (restart)",
                "S-RESYNCHRONIZE.request (set)",
                "S-RESYNCHRONIZE.response",
                "S-SYNCH-MAJOR.request",
                "S-SYNCH-MAJOR.response",
                "S-SYNCH-MINOR.request",
                "S-SYNCH-MINOR.response",
                "S-TYPED-DATA.request",
                "S-USER-ABORT.request",
                "S-USER-EXCEPTION-REPORT.request",
                "S-RECEIVE.request",
                "S-UNIT-DATA.request",                      /* 34 */
```

```
"illegal",                               /* 35 */
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",
"illegal",                               /* 89 */

"S-ACTIVITY-DISCARD.indication",         /* 90 */
"S-ACTIVITY-DISCARD.confirm",
"S-ACTIVITY-END.indication",
"S-ACTIVITY-END.confirm",
"S-ACTIVITY-INTERRUPT.indication",
"S-ACTIVITY-INTERRUPT.confirm",
"S-ACTIVITY-RESUME.indication",
"S-ACTIVITY-START.indication",
"S-CAPABILITY-DATA.indication",
"S-CAPABILITY-DATA.confirm",
"S-CONTROL-GIVE.indication",
"S-CONNECT.indication",
"S-CONNECT.confirm (accept)",
```

```
                "S-CONNECT.confirm (reject)",
                "S-DATA.indication",
                "S-DATA.confirm",
                "S-EXPEDITED-DATA.indication",
                "S-GIVE-TOKENS.indication",
                "S-PROVIDER-ABORT.indication",
                "S-PROVIDER-EXCEPTION-REPORT.indication",
                "S-PLEASE-TOKENS.indication",
                "S-RELEASE.indication",
                "S-RELEASE.confirm (accept)",
                "S-RELEASE.confirm (reject)",
                "S-RESYNCHRONIZE.indication",
                "S-RESYNCHRONIZE.confirm",
                "S-SYNCH-MAJOR.indication",
                "S-SYNCH-MAJOR.confirm",
                "S-SYNCH-MINOR.indication",
                "S-SYNCH-MINOR.confirm",
                "S-TYPED-DATA.indication",
                "S-USER-ABORT.indication",
                "S-USER-EXCEPTION-REPORT.indication",
                "S-UNIT-DATA.indication",
                "S-FLOW-CONTROL.indication"          /* 124, also SLAST */
            };

/*
 * Transport primitive names
 */

static char *Tevent[] = {
                "T-CONNECT.indication",      /* 0 */
                "T-CONNECT.confirm",         /* 1 */
                "T-DATA.indication",         /* 2 */
                "T-DISCONNECT.indication",   /* 3 */

                "T-CONNECT.request",         /* 4 */
                "T-CONNECT.response",        /* 5 */
                "T-DATA.request",            /* 6 */
                "T-DISCONNECT.request"       /* 7 */
            };

/*
 * Session Timer names
 */

static char *timer[] = {
                "Connect timer timeout",    /* 0 CONNECT_TIMER */
                "Abort timer timeout"       /* 1 ABORT_TIMER */
            };
```

```
/*
 * FUNCTION:  printHEX
 *
 *            This function prints out a string of 1-byte hexadecimal
 *            digits.
 *
 * INPUTS:    addr - start address of byte string.
 *            len  - length of byte string.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     printf.
 */

printHEX(addr,len)
uint8 *addr;
int   len;
{
  while (len--)
     .   printf("%.02X ",*addr++);
  printf("\n");
}

/*
 * FUNCTION:  printDEC
 *
 *            This function prints out a string of 1-byte decimal
 *            digits.
 *
 * INPUTS:    addr - start address of byte string.
 *            len  - length of byte string.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     printf.
 */

printDEC(addr,len)
uint8 *addr;
int   len;
{
  int   i;
  uint8 *maxaddr = addr + len;
  while (addr < maxaddr) {
        printf("   ");
        i = 1;
        while (addr < maxaddr && i++ <= 19)
              printf("%.03d ",*addr++);
        printf("\n");
  }
  printf("\n");
}
```

```
/*
 * FUNCTION:  printSPDU
 *
 *            This function prints out the contents of a given TSDU buffer
 *            as a string of 1-byte decimal digits.
 *
 * INPUTS:    tsdu - pointer to TSDU buffer.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     idSPDU, printf, printDEC.
 */

static void printSPDU(tsdu)
struct buf *tsdu;
{
 extern boolean idSPDU();
 printf("\n   The SPDU is ");
 if      (idSPDU(tsdu,CN))   printf("CONNECT:\n");
 else if (idSPDU(tsdu,AC))   printf("ACCEPT:\n");
 else if (idSPDU(tsdu,RFr))  printf("REFUSE (reuse):\n");
 else if (idSPDU(tsdu,RFnr)) printf("REFUSE (not reuse):\n");
 else if (idSPDU(tsdu,ABr))  printf("ABORT (reuse):\n");
 else if (idSPDU(tsdu,ABnr)) printf("ABORT (not reuse):\n");
 else if (idSPDU(tsdu,AA))   printf("ABORT ACCEPT:\n");
 printDEC(tsdu->addr,tsdu->length);
}
```

E.7 Session primitives source file listing: sprmtvs.c

```
/*
 * SESSION ind/cnf PRIMITIVE FUNCTIONS: sprmtvs.c
 * ED van der Westhuizen   September 1989
 *
 * FUNCTIONs:  Session indication/confirm primitives
 *
 *             The session entity calls these functions to pass session
 *             ind/cnf primitives to the SS-user. These functions then call
 *             the SS-user session ind/cnf primitive handler function,
 *             (*SSuser)(), which was received in an earlier SS-user call
 *             to s_activate().
 *
 * INPUTS:     see individual functions.
 *
 * OUTPUTS:    none.
 *
 * CALLS:      (*SSuser)(), bcopy.
 */

/*
 *  primitive:  S-CONNECT.indication
 */

static void SCONind(idu)
struct idu *idu;  /* SIDU */
{
 idu->event    = SCONIND;
 idu->rem_addr = &CURNTs->remSSAPaddr;
 bcopy(&defaultQOSS, &idu->qos, sizeof(qos_type));
 (*SSuser)(CURNTs,idu);

#ifdef DEBUG
  printf("   Output event: %s\n",Sevent[idu->event]);
  printf("          remote NSAPaddr: ");
  printHEX(idu->rem_addr->nsap.addr,idu->rem_addr->nsap.len);
  printf("          remote TSAPid:   ");
  printHEX(idu->rem_addr->tsap.addr,idu->rem_addr->tsap.len);
  printf("          remote SSAPid:   ");
  printHEX(idu->rem_addr->ssap.addr,idu->rem_addr->ssap.len);
#endif
}

/*
 *  primitive:  S-CONNECT.confirm (accept)
 */

static void SCONcnfacc(idu)
struct idu *idu; /* SIDU */
{
 idu->event  = SCONCNF;
 bcopy(&defaultQOSS, &idu->qos, sizeof(qos_type));
 (*SSuser)(CURNTs,idu);

#ifdef DEBUG
  printf("   Output event: %s\n",Sevent[idu->event]);
#endif
}
```

```
/*
 *  primitive:  S-CONNECT.confirm (reject)
 */

static void SCONcnfrej(idu)
struct idu *idu;  /* SIDU */
{
 idu->event  = SCONREF;
 idu->sn     = 0;
 idu->token  = (RT_INIT | MAT_INIT | SMT_INIT | DT_INIT);
 bcopy(&defaultQOSS, &idu->qos, sizeof(qos_type));
 (*SSuser)(CURNTs,idu);

#ifdef DEBUG
  printf("   Output event: %s\n",Sevent[idu->event]);
#endif
}

/*
 *  primitive:  S-DATA.confirm
 */

static void SDTcnf(buffer)
struct buf *buffer;  /* session layer data buffer */
{
 struct idu idu;
 idu.event  = SDTCNF;
 idu.buffer = buffer;
 (*SSuser)(CURNTs,&idu);

#ifdef DEBUG
  printf("   Output event: %s\n",Sevent[idu.event]);
#endif
}

/*
 *  primitive:  S-PROVIDER-ABORT.indication
 */

static void SPABind(reason)
uint8 reason;  /* abort reason */
{
 struct idu idu;
 idu.event  = SPABIND;
 idu.reason = reason;
 idu.buffer = NULL;
 (*SSuser)(CURNTs,&idu);

#ifdef DEBUG
  printf("   Output event: %s\n",Sevent[idu.event]);
#endif
}

/*
 *  primitive:  S-U-ABORT.indication
 */

static void SUABind(buffer)
struct buf *buffer;  /* session layer data buffer */
{
 struct idu idu;
 idu.event  = SUABIND;
 idu.buffer = buffer;
 (*SSuser)(CURNTs,&idu);

#ifdef DEBUG
  printf("   Output event: %s\n",Sevent[idu.event]);
#endif
}
```

E.8 Transport primitives source file listing: tprmtvs.c

```
/*
 * TRANSPORT req/rsp PRIMITIVE FUNCTIONS: tprmtvs.c
 * ED van der Westhuizen   September 1989
 *
 * FUNCTIONs:  Transport request/response primitives
 *
 *             The TS-user calls these functions to pass transport req/rsp
 *             primitives to the transport entity. These functions then
 *             call appropriate transport req/rsp handlers in the transport
 *             entity itself.
 *
 * INPUTS:     see individual functions.
 *
 * OUTPUTS:    If TCONreq is successful, it returns the TCEPid, otherwise
 *             it returns NULL.
 *
 * CALLS:      Transport request/response primitive handlers:
 *             UCONreq, UCONres, UDATreq, UDISreq.
 */

#ifdef DEBUG
#define TCONREQ  4  /* T-CONNECT.request */
#define TCONRSP  5  /* T-CONNECT.response */
#define TDTREQ   6  /* T-DATA.request */
#define TDISREQ  7  /* T-DISCONNECT.request */
#endif

/*
 *  primitive:  T-CONNECT.request
 */

static pointer TCONreq(cldNSAPaddr,cldTSAPid)
nsap_address  *cldNSAPaddr;      /* called NSAP address */
tsap_selector *cldTSAPid;        /* called TSAP id */
{
 pointer retval;
 retval = UCONreq(&locTSAPid,    /* calling TSAP id */
                  cldNSAPaddr,   /* called NSAP address */
                  cldTSAPid,     /* called TSAPid */
                  &defaultQOTS,  /* proposed QOTS */
                  FALSE,         /* proposed TEXP option */
                  NULL);         /* TS-user data */

#ifdef DEBUG
  printf("   Output event: %s ",Tevent[TCONREQ]);
  if (retval) printf("(successful)\n"); else printf("(unsuccessful)\n");
#endif

 return(retval);
}
```

```
/*
 *  primitive:  T-CONNECT.response
 */

static void TCONrsp(TCEPid)
pointer TCEPid;          /* TCEPid */
{
 UCONres(TCEPid,         /* TCEPid */
         &defaultQOTS,   /* selected QOTS */
         FALSE,          /* selected TEXP option */
         NULL);          /* TS-user data */

#ifdef DEBUG
  printf("   Output event: %s\n",Tevent[TCONRSP]);
#endif
}

/*
 *  primitive:  T-DATA.request
 */

static void TDTreq(TCEPid,tsdu)
pointer    TCEPid;  /* TCEPid */
struct buf *tsdu;   /* pointer to TSDU buffer */
{
 UDATreq(TCEPid,    /* TCEPid */
         tsdu,      /* TSDU buffer */
         TRUE);     /* eotsdu flag */

#ifdef DEBUG
  printf("   Output event: %s\n",Tevent[TDTREQ]);
  printSPDU(tsdu);
#endif
}

/*
 *  primitive:  T-DISCONNECT.request
 */

static void TDISreq(TCEPid)
pointer TCEPid;  /* TCEPid */
{
 UDISreq(TCEPid, /* TCEP id */
         NULL);  /* TS-user data */

#ifdef DEBUG
  printf("   Output event: %s\n",Tevent[TDISREQ]);
#endif
}

#ifdef DEBUG
#undef TCONREQ
#undef TCONRSP
#undef TDTREQ
#undef TDISREQ
#endif
```

E.9 Miscellaneous functions source file listing: funcs1.c

```
/*
 * SESSION ENTITY STATIC FUNCTIONS: funcs1.c
 * ED van der Westhuizen   September 1989
 *
 * function          calls
 * --------          -----
 * ClockInterrupt    os_sigset, os_alarm, clock, do_timer_queue.
 * FU                none.
 * AV                FU.
 * p                 none.
 * SpAc              FU, cantimer, newtimer, SessionTimeOut.
 */

/*
 * FUNCTION:  ClockInterrupt
 *
 *            This function provides a periodic (1 sec) clock interrupt
 *            for the timer manager module.
 *
 *            To start this interrupt mechanism, this function is called
 *            once from init_session(). Thereafter, it re-triggers itself
 *            on interrupt.
 *
 *            This function calls timer module function clock() to
 *            increment real time, and then calls timer module function
 *            do_timer_queue() to process any expired timers.
 *
 * INPUTS:    none.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     os_sigset, os_alarm, clock, do_timer_queue.
 */

static void ClockInterrupt()
{
 os_sigset(SIGALRM,ClockInterrupt);  /* call ClockInterrupt on SIGALRM */
 os_alarm(CLOCK/1000);               /* receive SIGALRM after 1 sec */
 clock();                            /* increment real time */
 do_timer_queue();                   /* process expired timers */
}

/*
 * FUNCTION:  FU
 *
 *            This function tests whether a given functional unit
 *            has been selected for use on a session connection.
 *
 * INPUTS:    fu - the functional unit.
 *
 * OUTPUTS:   returns: TRUE  if the functional unit has been selected,
 *                     FALSE if not.
 *
 * CALLS:     none.
 */

static boolean FU(fu)
uint16 fu;
{
 return((fu & CURNTs->fus) ? TRUE : FALSE);
}
```

```
/*
 * FUNCTION:  AV
 *
 *            This function tests whether a given token
 *            is available for use on a session connection.
 *
 * INPUTS:    token - the token.
 *
 * OUTPUTS:   returns: TRUE  if the token is available,
 *                     FALSE if not.
 *
 * CALLS:     FU.
 */

static boolean AV(token)
uint8 token;
{
 switch (token) {
 case MIT: return(FU(MIS));
 case DKT: return(FU(HDX));
 case TRT: return(FU(NRL));
 case MAT: return((FU(MAS) || FU(ACT)) ? TRUE : FALSE);
 }
 /* NOTREACHED */
}

/*
 * FUNCTION:  p
 *
 *            This function implements the boolean predicate conditions
 *            as specified in Rec. X.225: TABLE A-6/X.225.
 *
 * INPUTS:    n - the predicate selector.
 *
 * OUTPUTS:   returns: TRUE  if the predicate is true,
 *                     FALSE if not.
 *
 * CALLS:     none.
 */

static boolean p(n)
int n;
{
 switch (n) {
 case 1:
      return(!CURNTs->Vtca);
 case 2:
      return(REUSE_TC && !CURNTs->Texp);
 default:
      return(FALSE);
 }
}
```

```
/*
 * FUNCTION:  SpAc
 *
 *            This function implements the specific actions as
 *            specified in Rec. X.225: TABLE A-5/X.225.
 *
 * INPUTS:    n      - the specific action selector.
 *            tokens - currently owned tokens,    SpAc(11) only.
 *            sn     - initial sync number,       SpAc(5) only.
 *            texp   - selected TEXP option,      SpAc(5) only.
 *            fus    - selected functional units, SpAc(5) only.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     FU, cantimer, newtimer, SessionTimeOut.
 */

static void SpAc(n,tokens,sn,texp,fus)
int     n;
uint8   tokens;
uint32  sn;
boolean texp;
uint16  fus;
{
 extern void SessionTimeOut();
 switch (n) {
 case 1:
      CURNTs->Vtca = TRUE;
      break;

 case 2:
      CURNTs->Vtca = FALSE;
      break;

 case 3:
      cantimer(CURNTs,ABORT_TIMER,0);
      break;

 case 4:
      newtimer(CURNTs,ABORT_TIMER,FASTTIMER,0,0,SessionTimeOut);
      break;

 case 5:
      CURNTs->Va    = sn;
      CURNTs->Vm    = sn;
      CURNTs->Vsc   = FALSE;
      CURNTs->Texp  = texp;
      CURNTs->fus   = fus;
      if (FU(ACT))
         CURNTs->Vact = FALSE;
      break;

 case 11:
      CURNTs->AsTokens = tokens;
      break;

 case 32:
      cantimer(CURNTs,CONNECT_TIMER,0);
      break;

 case 33:
      newtimer(CURNTs,CONNECT_TIMER,SLOWTIMER,0,0,SessionTimeOut);
      break;

 default:
      break;
 }
}
```

E.10 TSDU stripping functions source file listing: strip.c

```
/*
 * TSDU STRIPPING FUNCTIONS: strip.c
 * ED van der Westhuizen   September 1989
 *
 * These functions are used to strip (from a TSDU buffer):
 *   SPDUs from TSDUs,
 *   PGIUs and PIUs from SPDUs,
 *   PIUs from PGIUs,
 *   parameter values from PIUs.
 *
 * Each unit (TSDU, SPDU, PGIU, PIU) is stripped from its 1st
 * byte, in the direction of the end of the TSDU.
 * None of these functions employ any structure error checking.
 */

/*
 * FUNCTION:  StripHeader
 *
 *            This function strips a PIU/PGIU/SPDU header from a
 *            a given TSDU buffer.
 *
 * INPUTS:    tsdu  - a pointer to the TSDU buffer. The .addr and
 *                    .length fields indicate the current TSDU size.
 *            Pcode - the PI/PGI/SI code of the required PIU/PGIU/SPDU.
 *                    If Pcode is 0, the header is stripped irrespective
 *                    of the actual PI/PGI/SI value.
 *
 * OUTPUTS:   returns: the PIU/PGIU/SPDU LI value.
 *              LI will = 0 if:
 *                 a) The end of the TSDU has been reached, OR
 *                 b) The PIU/PGIU/SPDU is not the required one, OR
 *                 c) The LI field contains the value 0.
 *              In this event, there are no parameters for the
 *              required PIU/PGIU/SPDU present in the TSDU.
 *
 *            The TSDU buffer .addr and .length fields are updated to
 *            exclude the stripped header, if it was present.
 *
 * CALLS:     get1, get2
 */

static uint16 StripHeader(tsdu,Pcode)
struct buf *tsdu;
uint8      Pcode;
{
 uint16 LI = 0;                    /* length indicator */
 if (tsdu->length  > 0     &&      /* if not end of TSDU AND   */
     (*tsdu->addr == Pcode ||      /* if required PI/PGI/SI OR */
           Pcode == 0       )) { /*         any PI/PGI/SI    */
    tsdu->addr++;                  /* point to 1st LI byte */
    get1(LI,tsdu->addr);           /* get 1 byte LI */
    tsdu->length -= 2;
    if (LI == 255) {               /* if 3 byte LI field */
       get2(LI,tsdu->addr);        /* get 2 byte LI */
       tsdu->length -= 2;
    }
 }
 return(LI);
}
```

```
/*
 * FUNCTION:  PIU stripping functions
 *
 *            These functions each strip a particular PIU and its
 *            parameter value from a given TSDU buffer. The parameter
 *            value is placed in a variable pointed to by a given pointer.
 *            Since the inclusion of certain PIUs in a SPDU is
 *            non-mandatory, these functions assign appropriate
 *            default values to required PIU parameters not present in
 *            the SPDU.
 *
 * INPUTS:    tsdu      - a pointer to the TSDU buffer. The .addr and
 *                        .length fields indicate the current TSDU size.
 *            parameter - a pointer a data structure to hold the stripped
 *                        PIU parameter.
 *
 * OUTPUTS:   *parameter - holds the stripped PIU paremeter.
 *
 *            The TSDU buffer .addr and .length fields are updated to
 *            exclude the stripped PIU, if it was present.
 *
 * CALLS:     StripHeader, get1, get2, gets
 */

/*
 * Strip Called SS-user reference
 */
static void Strip9PIU(tsdu,scid)
struct buf  *tsdu;
struct scid *scid;
{
 uint16 LI;
 scid->cld_ref.len = 0;
 if (LI = StripHeader(tsdu,9)) {
    gets(tsdu->addr,scid->cld_ref.data,LI);
    scid->cld_ref.len = LI;
    tsdu->length -= LI;
 }
}

/*
 * Strip Calling SS-user reference
 */
static void Strip10PIU(tsdu,scid)
struct buf  *tsdu;
struct scid *scid;
{
 uint16 LI;
 scid->clg_ref.len = 0;
 if (LI = StripHeader(tsdu,10)) {
    gets(tsdu->addr,scid->clg_ref.data,LI);
    scid->clg_ref.len = LI;
    tsdu->length -= LI;
 }
}
```

```
/*
 * Strip Common reference
 */
static void Strip11PIU(tsdu,scid)
struct buf  *tsdu;
struct scid *scid;
{
 uint16 LI;
 scid->com_ref.len = 0;
 if (LI = StripHeader(tsdu,11)) {
    gets(tsdu->addr,scid->com_ref.data,LI);
    scid->com_ref.len = LI;
    tsdu->length -= LI;
 }
}

/*
 * Strip Additional reference information
 */
static void Strip12PIU(tsdu,scid)
struct buf  *tsdu;
struct scid *scid;
{
 uint16 LI;
 scid->add_ref.len = 0;
 if (LI = StripHeader(tsdu,12)) {
    gets(tsdu->addr,scid->add_ref.data,LI);
    scid->add_ref.len = LI;
    tsdu->length -= LI;
 }
}

/*
 * Strip Token item
 */
static void Strip16PIU(tsdu,TokenItem)
struct buf *tsdu;
uint8      *TokenItem;
{
 *TokenItem = 0;
 if (StripHeader(tsdu,16)) {
    get1(*TokenItem,tsdu->addr);
    tsdu->length--;
 }
}

/*
 * Strip Transport disconnect
 */
static void Strip17PIU(tsdu,TCdis)
struct buf   *tsdu;
struct TCdis *TCdis;
{
 uint8  byte;
 TCdis->TCkept   = FALSE;
 TCdis->ABreason = NO_REASON;
 if (StripHeader(tsdu,17)) {
    get1(byte,tsdu->addr);
    TCdis->ABreason = byte & 0xFE;
    TCdis->TCkept   = ((byte & 0x01) ? FALSE : TRUE);
    tsdu->length--;
 }
}
```

```
/*
 * Strip Protocol options
 */
static void Strip19PIU(tsdu,protocol)
struct buf *tsdu;
uint8      *protocol;
{
 *protocol = 0;
 if (StripHeader(tsdu,19)) {
    get1(*protocol,tsdu->addr);
    tsdu->length--;
 }
}

/*
 * Strip Session user requirements
 */
static void Strip20PIU(tsdu,fus)
struct buf *tsdu;
uint16     *fus;
{
 *fus = FU_SUP;
 if (StripHeader(tsdu,20)) {
    get2(*fus,tsdu->addr);
    tsdu->length -= 2;
 }
}

/*
 * Strip TSDU maximum size
 */
static void Strip21PIU(tsdu,max0TSDUlen,max1TSDUlen)
struct buf *tsdu;
uint16     *max0TSDUlen,*max1TSDUlen;
{
 *max0TSDUlen = 0;
 *max1TSDUlen = 0;
 if (StripHeader(tsdu,21)) {
    get2(*max0TSDUlen,tsdu->addr);
    get2(*max1TSDUlen,tsdu->addr);
    tsdu->length -= 4;
 }
}

/*
 * Strip Version number
 */
static void Strip22PIU(tsdu,version)
struct buf *tsdu;
uint8      *version;
{
 *version = VERSION;
 if (StripHeader(tsdu,22)) {
    get1(*version,tsdu->addr);
    tsdu->length--;
 }
}
```

```
/*
 * Strip Initial serial number
 */
static void Strip23PIU(tsdu,InitialSpsn)
struct buf *tsdu;
uint32     *InitialSpsn;
{
 uint16 LI;
 *InitialSpsn = 0;
 if (LI = StripHeader(tsdu,23)) {
    uint8  digit;
    uint32 factor = 1;
    uint16 i      = LI;
    *InitialSpsn  = 0;
    while (i-- > 1)
          factor *= 10;
    for (i = 1; i <= LI; i++) {
        get1(digit,tsdu->addr);
        *InitialSpsn += (digit - 48) * factor;
        factor /= 10;
    }
    tsdu->length -= LI;
 }
}

/*
 * Strip Token setting item
 */
static void Strip26PIU(tsdu,TokenSetItem)
struct buf *tsdu;
uint8      *TokenSetItem;
{
 *TokenSetItem = (RT_RESP | MAT_RESP | SMT_RESP | DT_RESP);
 if (StripHeader(tsdu,26)) {
    get1(*TokenSetItem,tsdu->addr);
    tsdu->length--;
 }
}

/*
 * Strip Reflect parameter values
 */
static void Strip49PIU(tsdu,ReflectParams)
struct buf    *tsdu;
struct Bytes9 *ReflectParams;
{
 uint16 LI;
 ReflectParams->len = 0;
 if (LI = StripHeader(tsdu,49)) {
    gets(tsdu->addr,ReflectParams->data,LI);
    ReflectParams->len = LI;
    tsdu->length -= LI;
 }
}
```

```
/*
 * Strip Reason code
 */
static void Strip50PIU(tsdu,ReasonCode,SSuserData)
struct buf *tsdu;
uint8      *ReasonCode;
struct buf **SSuserData;
{
 *ReasonCode = SSU_UNSPECIFIED;
 *SSuserData = NULL;
 if (StripHeader(tsdu,50)) {
    get1(*ReasonCode,tsdu->addr);
    tsdu->length--;
    *SSuserData = ((tsdu->length) ? tsdu : NULL);
 }
}

/*
 * Strip Calling SSAP identifier
 */
static void Strip51PIU(tsdu,clgSSAPid)
struct buf    *tsdu;
ssap_selector *clgSSAPid;
{
 uint16 LI;
 clgSSAPid->len = 0;
 if (LI = StripHeader(tsdu,51)) {
    gets(tsdu->addr,clgSSAPid->addr,LI);
    clgSSAPid->len = LI;
    tsdu->length -= LI;
 }
}

/*
 * Strip Called SSAP identifier
 */
static void Strip52PIU(tsdu,cldSSAPid)
struct buf    *tsdu;
ssap_selector *cldSSAPid;
{
 uint16 LI;
 cldSSAPid->len = 0;
 if (LI = StripHeader(tsdu,52)) {
    gets(tsdu->addr,cldSSAPid->addr,LI);
    cldSSAPid->len = LI;
    tsdu->length -= LI;
 }
}
```

```
/*
 * FUNCTION:  PGIU stripping functions
 *
 *            These functions each strip a particular PGIU and its
 *            parameter values from a given TSDU buffer. The parameter
 *            values are placed in variables pointed to by given pointers.
 *
 * INPUTS:    tsdu       - a pointer to the TSDU buffer. The .addr and
 *                         .length fields indicate the current TSDU size.
 *            parameters - pointers to data structures to hold
 *                         the stripped PGIU parameters.
 *
 * OUTPUTS:   *parameters - hold the stripped PGIU parameters.
 *
 *            The TSDU buffer .addr and .length fields are updated to
 *            exclude the stripped PGIU, if it was present.
 *
 * CALLS:     StripHeader, PIU stripping functions
 */

/*
 * Strip Connection identifier for CN SPDU
 */
static void Strip1aPGIU(tsdu,scid)
struct buf  *tsdu;
struct scid *scid;
{
 StripHeader(tsdu,1);
 Strip10PIU(tsdu,scid);
 Strip11PIU(tsdu,scid);
 Strip12PIU(tsdu,scid);
}

/*
 * Strip Connection identifier for AC,RF SPDUs
 */
static void Strip1bPGIU(tsdu,scid)
struct buf  *tsdu;
struct scid *scid;
{
 StripHeader(tsdu,1);
 Strip9PIU(tsdu,scid);
 Strip11PIU(tsdu,scid);
 Strip12PIU(tsdu,scid);
}

/*
 * Strip Connect/Accept item
 */
static void Strip5PGIU(tsdu,protocol,
                       max0TSDUlen, max1TSDUlen,
                       version,
                       InitialSpsn,
                       TokenSetItem)
struct buf *tsdu;
uint8      *protocol;
uint16     *max0TSDUlen, *max1TSDUlen;
uint8      *version;
uint32     *InitialSpsn;
uint8      *TokenSetItem;
{
 StripHeader(tsdu,5);
 Strip19PIU(tsdu,protocol);
 Strip21PIU(tsdu,max0TSDUlen,max1TSDUlen);
 Strip22PIU(tsdu,version);
 Strip23PIU(tsdu,InitialSpsn);
 Strip26PIU(tsdu,TokenSetItem);
}
```

```
/*
 * Strip User data
 */
static void Strip193PGIU(tsdu,SSuserData)
struct buf *tsdu;
struct buf **SSuserData;
{
 *SSuserData = NULL;
 if (StripHeader(tsdu,193))
    *SSuserData = ((tsdu->length) ? tsdu : NULL);
}

/*
 * FUNCTION:  SPDU stripping functions
 *
 *            These functions each strip a particular SPDU and its
 *            parameter values from a given TSDU buffer. The parameter
 *            values are placed in variables pointed to by given pointers.
 *            To strip concatenated SPDUs from a TSDU buffer, call the
 *            appropriate SPDU stripping functions in order.
 *
 * INPUTS:    tsdu       - a pointer to the TSDU buffer. The .addr and
 *                         .length fields indicate the current TSDU size.
 *            parameters - pointers to data structures to hold the
 *                         stripped SPDU parameters.
 *
 * OUTPUTS:   *parameters - hold the stripped SPDU parameters.
 *
 *            The TSDU buffer .addr and .length fields are updated to
 *            exclude the stripped SPDU, if it was present.
 *
 * CALLS:     StripHeader, PIU and PGIU srtripping functions
 */

/*
 * Strip the REFUSE SPDU
 */
static void StripRF(tsdu,scid,
                    TCdis,
                    Srequirements,
                    version,
                    ReasonCode,
                    SSuserData)
struct buf   *tsdu;
struct scid  *scid;
struct TCdis *TCdis;
uint16       *Srequirements;
uint8        *version;
uint8        *ReasonCode;
struct buf   **SSuserData;
{
 StripHeader(tsdu,12);
 Strip1bPGIU(tsdu,scid);
 Strip17PIU(tsdu,TCdis);
 Strip20PIU(tsdu,Srequirements);
 Strip22PIU(tsdu,version);
 Strip50PIU(tsdu,ReasonCode,SSuserData);
}
```

```
/*
 * Strip the CONNECT SPDU
 */
static void StripCN(tsdu,scid,
                    protocol,
                    max0TSDUlen,
                    max1TSDUlen,
                    version,
                    InitialSpsn,
                    TokenSetItem,
                    fus,
                    clgSSAPid,
                    cldSSAPid,
                    SSuserData)
struct buf     *tsdu;
struct scid    *scid;
uint8          *protocol;
uint16         *max0TSDUlen;
uint16         *max1TSDUlen;
uint8          *version;
uint32         *InitialSpsn;
uint8          *TokenSetItem;
uint16         *fus;
ssap_selector *clgSSAPid;
ssap_selector *cldSSAPid;
struct buf     **SSuserData;
{
 StripHeader(tsdu,13);
 StriplaPGIU(tsdu,scid);
 Strip5PGIU(tsdu,protocol,
            max0TSDUlen,
            max1TSDUlen,
            version,
            InitialSpsn,
            TokenSetItem);
 Strip20PIU(tsdu,fus);
 Strip51PIU(tsdu,clgSSAPid);
 Strip52PIU(tsdu,cldSSAPid);
 Strip193PGIU(tsdu,SSuserData);
}
```

```
/*
 * Strip the ACCEPT SPDU
 */
static void StripAC(tsdu,scid,
                    protocol,
                    max0TSDUlen,
                    max1TSDUlen,
                    version,
                    InitialSpsn,
                    TokenSetItem,
                    TokenItem,
                    Srequirements,
                    clgSSAPid,
                    cldSSAPid,
                    SSuserData)
struct buf    *tsdu;
struct scid   *scid;
uint8         *protocol;
uint16        *max0TSDUlen;
uint16        *max1TSDUlen;
uint8         *version;
uint32        *InitialSpsn;
uint8         *TokenSetItem;
uint8         *TokenItem;
uint16        *Srequirements;
ssap_selector *clgSSAPid;
ssap_selector *cldSSAPid;
struct buf    **SSuserData;
{
 StripHeader(tsdu,14);
 Strip1bPGIU(tsdu,scid);
 Strip5PGIU(tsdu,protocol,
                max0TSDUlen,
                max1TSDUlen,
                version,
                InitialSpsn,
                TokenSetItem);
 Strip16PIU(tsdu,TokenItem);
 Strip20PIU(tsdu,Srequirements);
 Strip51PIU(tsdu,clgSSAPid);
 Strip52PIU(tsdu,cldSSAPid);
 Strip193PGIU(tsdu,SSuserData);
}

/*
 * Strip the ABORT SPDU
 */
static void StripAB(tsdu,TCdis,
                    ReflectParams,
                    SSuserData)
struct buf    *tsdu;
struct TCdis  *TCdis;
struct Bytes9 *ReflectParams;
struct buf    **SSuserData;
{
 StripHeader(tsdu,25);
 Strip17PIU(tsdu,TCdis);
 Strip49PIU(tsdu,ReflectParams);
 Strip193PGIU(tsdu,SSuserData);
}
```

E.11 TSDU building functions source file listing: build.c

```
/*
 * TSDU BUILDING FUNCTIONS: build.c
 * ED van der Westhuizen   September 1989
 *
 * These functions are used to construct (in a TSDU buffer):
 *   TSDUs from SPDUs,
 *   SPDUs from PGIUS and PIUs,
 *   PGIUs from PIUs,
 *   PIUs  from parameter values.
 *
 * Each unit (TSDU, SPDU, PGIU, PIU) is built
 * from its last byte, in the direction of the beginning of the TSDU.
 */

#define ZERO ((unsigned) 0)  /* for unsigned comparisons with 0 */

/*
 * FUNCTION:   BuildHeader
 *
 *             This function prepends a PIU/PGIU/SPDU header onto
 *             a given TSDU buffer.
 *
 * INPUTS:     tsdu  -  a pointer to the TSDU buffer. The .addr and
 *                      .length fields indicate the current TSDU size.
 *             Pcode - the PI/PGI/SI code.
 *             LI    - the Length Indicator.
 *
 * OUTPUTS:    The TSDU buffer .addr and .length fields are updated to
 *             include the prepended header.
 *
 * CALLS:      add1, add2
 */

static void BuildHeader(tsdu,Pcode,LI)
struct buf *tsdu;
uint8      Pcode;
uint16     LI;
{
 uint8 *bp;
 tsdu->length += ((LI>254) ? 4 : 2);     /* update tsdu length field */
 bp = tsdu->addr -= ((LI>254) ? 4 : 2); /* update tsdu addr field */
 add1(bp,Pcode);                         /* build PI/PGI/SI field */
 if (LI > 254) {
    add1(bp,255);                        /* build 3 byte LI field */
    add2(bp,LI);
 }
 else
    add1(bp,LI);                         /* build 1 byte LI field */
}
```

```
/*
 * FUNCTION:  PIU building functions
 *
 *            Given the appropriate parameter, each of these functions
 *            prepends a particular PIU onto a given TSDU buffer.
 *            The inclusion of certain PIUs in a SPDU is non-mandatory,
 *            and therefore depends on certain conditions.
 *            These functions only prepend PIUs onto the TSDU if
 *            the relevant conditions are satisfied.
 *
 * INPUTS:    tsdu      - a pointer to the TSDU buffer. The .addr and
 *                        .length fields indicate the current TSDU size.
 *            parameter - the PIU parameter.
 *
 * OUTPUTS:   The TSDU buffer .addr and .length fields are updated to
 *            include the prepended PIU.
 *
 * CALLS:     BuildHeader FU
 *            add1 add2 adds
 */

/*
 * Build Called SS-user reference
 */
static void Build9PIU(tsdu,scid)
struct buf  *tsdu;
struct scid *scid;
{
 uint8  *bp;
 uint8  PI = 9;
 uint16 LI = scid->cld_ref.len;
 if (LI > ZERO && LI <= 64) {
    tsdu->length += LI;
    bp = tsdu->addr -= LI;
    adds(bp,scid->cld_ref.data,LI);
    BuildHeader(tsdu,PI,LI);
 }
}

/*
 * Build Calling SS-user reference
 */
static void Build10PIU(tsdu,scid)
struct buf  *tsdu;
struct scid *scid;
{
 uint8  *bp;
 uint8  PI = 10;
 uint16 LI = scid->clg_ref.len;
 if (LI > ZERO && LI <= 64) {
    tsdu->length += LI;
    bp = tsdu->addr -= LI;
    adds(bp,scid->clg_ref.data,LI);
    BuildHeader(tsdu,PI,LI);
 }
}
```

```
/*
 * Build Common reference
 */
static void Build11PIU(tsdu,scid)
struct buf  *tsdu;
struct scid *scid;
{
 uint8  *bp;
 uint8  PI = 11;
 uint16 LI = scid->com_ref.len;
 if (LI > ZERO && LI <= 64) {
    tsdu->length += LI;
    bp = tsdu->addr -= LI;
    adds(bp,scid->com_ref.data,LI);
    BuildHeader(tsdu,PI,LI);
 }
}

/*
 * Build Additional reference information
 */
static void Build12PIU(tsdu,scid)
struct buf  *tsdu;
struct scid *scid;
{
 uint8  *bp;
 uint8  PI = 12;
 uint16 LI = scid->add_ref.len;
 if (LI > ZERO && LI <= 4) {
    tsdu->length += LI;
    bp = tsdu->addr -= LI;
    adds(bp,scid->add_ref.data,LI);
    BuildHeader(tsdu,PI,LI);
 }
}

/*
 * Build Token item
 */
static void Build16PIU(tsdu,TokenItem)
struct buf *tsdu;
uint8      TokenItem;
{
 uint8  *bp;
 uint8  PI = 16;
 uint16 LI = 1;
 if (TokenItem != ZERO) {
    tsdu->length += LI;
    bp = tsdu->addr -= LI;
    add1(bp,TokenItem);
    BuildHeader(tsdu,PI,LI);
 }
}
```

```
/*
 * Build Transport disconnect
 */
static void Build17PIU(tsdu,TCdis)
struct buf   *tsdu;
struct TCdis *TCdis;
{
 uint8 *bp;
 uint8  PI = 17;
 uint16 LI = 1;
 if (TRUE) {
    tsdu->length += LI;
    bp = tsdu->addr -= LI;
    add1(bp,((TCdis->TCkept) ? 0 : 1) + TCdis->ABreason);
    BuildHeader(tsdu,PI,LI);
 }
}

/*
 * Build Protocol options
 */
static void Build19PIU(tsdu,protocol)
struct buf *tsdu;
uint8      protocol;
{
 uint8 *bp;
 uint8  PI = 19;
 uint16 LI = 1;
 if (TRUE) {
    tsdu->length += LI;
    bp = tsdu->addr -= LI;
    add1(bp,protocol);
    BuildHeader(tsdu,PI,LI);
 }
}

/*
 * Build Session user requirements
 */
static void Build20PIU(tsdu,fus)
struct buf *tsdu;
uint16     fus;
{
 uint8 *bp;
 uint8  PI = 20;
 uint16 LI = 2;
 if (TRUE) {
    tsdu->length += LI;
    bp = tsdu->addr -= LI;
    add2(bp,fus);
    BuildHeader(tsdu,PI,LI);
 }
}
```

```
/*
 * Build TSDU maximum size
 */
static void Build21PIU(tsdu,max0TSDUlen,max1TSDUlen)
struct buf *tsdu;
uint16     max0TSDUlen,max1TSDUlen;
{
 uint8  *bp;
 uint8  PI = 21;
 uint16 LI = 4;
 if (max0TSDUlen > ZERO || max1TSDUlen > ZERO) {
    tsdu->length += LI;
    bp = tsdu->addr -= LI;
    add2(bp,max0TSDUlen);
    add2(bp,max1TSDUlen);
    BuildHeader(tsdu,PI,LI);
 }
}

/*
 * Build Version number
 */
static void Build22PIU(tsdu,version)
struct buf *tsdu;
uint8      version;
{
 uint8 *bp;
 uint8  PI = 22;
 uint16 LI = 1;
 if (TRUE) {
    tsdu->length += LI;
    bp = tsdu->addr -= LI;
    add1(bp,version);
    BuildHeader(tsdu,PI,LI);
 }
}

/*
 * Build Initial serial number
 */
static void Build23PIU(tsdu,sn)
struct buf *tsdu;
uint32     sn;
{
 uint8  *bp;
 uint8  PI = 23;
 uint16 LI;
 uint32 factor;
 if (!FU(ACT) && (FU(MIS) || FU(MAS) || FU(RES))) {
    for (factor=100000, LI=6; sn/factor==0 && LI>1; factor/=10, LI--) ;
    tsdu->length += LI;
    bp = tsdu->addr -= LI;
    for (; LI > ZERO; factor /= 10, LI--) {
        add1(bp,sn/factor + 48);
        sn %= factor;
    }
    BuildHeader(tsdu,PI,LI);
 }
}
```

```
/*
 * Build Token setting item
 */
static void Build26PIU(tsdu,TokenSetItem)
struct buf *tsdu;
uint8      TokenSetItem;
{
 uint8   *bp;
 uint8   PI = 26;
 uint16  LI = 1;
 boolean SomeAvail = FALSE;
 uint8   token;
 for (token = DKT; token & TDM && !SomeAvail; token <<= 2)
     SomeAvail = FU(token);
 if (SomeAvail) {
    tsdu->length += LI;
    bp = tsdu->addr -= LI;
    add1(bp,TokenSetItem);
    BuildHeader(tsdu,PI,LI);
 }
}

/*
 * Build Reflect parameter values
 */
static void Build49PIU(tsdu,ReflectParams)
struct buf    *tsdu;
struct Bytes9 *ReflectParams;
{
 uint8  *bp;
 uint8  PI = 49;
 uint16 LI;
 if (ReflectParams && (LI = ReflectParams->len)) {
    tsdu->length += LI;
    bp = tsdu->addr -= LI;
    adds(bp,ReflectParams->data,LI);
    BuildHeader(tsdu,PI,LI);
 }
}

/*
 * Build Reason code
 */
static void Build50PIU(tsdu,ReasonCode)
struct buf *tsdu;
uint8      ReasonCode;
{
 uint8  *bp;
 uint8  PI = 50;
 uint16 LI = tsdu->length + 1;
 if (TRUE) {
    /* the (512 max) data bytes of ReasonCode are already in tsdu */
    tsdu->length += 1;
    bp = tsdu->addr -= 1;
    addl(bp,ReasonCode);
    BuildHeader(tsdu,PI,LI);
 }
}
```

```
/*
 * Build Calling SSAP identifier
 */
static void Build51PIU(tsdu,clgSSAPid)
struct buf    *tsdu;
ssap_selector *clgSSAPid;
{
 uint8  *bp;
 uint8  PI = 51;
 uint16 LI = clgSSAPid->len;
 if (LI > ZERO && LI <= 16) {
    tsdu->length += LI;
    bp = tsdu->addr -= LI;
    adds(bp,clgSSAPid->addr,LI);
    BuildHeader(tsdu,PI,LI);
 }
}

/*
 * Build Called SSAP identifier
 */
static void Build52PIU(tsdu,cldSSAPid)
struct buf    *tsdu;
ssap_selector *cldSSAPid;
{
 uint8  *bp;
 uint8  PI = 52;
 uint16 LI = cldSSAPid->len;
 if (LI > ZERO && LI <= 16) {
    tsdu->length += LI;
    bp = tsdu->addr -= LI;
    adds(bp,cldSSAPid->addr,LI);
    BuildHeader(tsdu,PI,LI);
 }
}

/*
 * FUNCTION:  PGIU building functions
 *
 *            Given the appropriate parameters, each of these functions
 *            prepends a particular PGIU onto a given TSDU buffer.
 *
 * INPUTS:    tsdu       - a pointer to the TSDU buffer. The .addr and
 *                         .length fields indicate the current TSDU size.
 *            parameters - the PGIU parameters.
 *
 * OUTPUTS:   The TSDU buffer .addr and .length fields are updated to
 *            include the prepended PGIU.
 *
 * CALLS:     BuildHeader,  PIU building functions
 */

/*
 * Build Connection identifier for CN SPDU
 */
static void Build1aPGIU(tsdu,scid)
struct buf  *tsdu;
struct scid *scid;
{
 uint8  PGI = 1;
 uint16 len1,len2,LI;
 len1 = tsdu->length;
 Build12PIU(tsdu,scid);
 Build11PIU(tsdu,scid);
 Build10PIU(tsdu,scid);
 len2 = tsdu->length;
 if ((LI = len2 - len1) > ZERO)
    BuildHeader(tsdu,PGI,LI);
}
```

```
/*
 * Build Connection identifier for AC,RF SPDU
 */
static void Build1bPGIU(tsdu,scid)
struct buf  *tsdu;
struct scid *scid;
{
 uint8  PGI = 1;
 uint16 len1,len2,LI;
 len1 = tsdu->length;
 Build12PIU(tsdu,scid);
 Build11PIU(tsdu,scid);
 Build9PIU(tsdu,scid);
 len2 = tsdu->length;
 if ((LI = len2 - len1) > ZERO)
    BuildHeader(tsdu,PGI,LI);
}

/*
 * Build Connect/Accept item
 */
static void Build5PGIU(tsdu,protocol,
                       max0TSDUlen,
                       max1TSDUlen,
                       version,
                       InitialSpsn,
                       TokenSetItem)
struct buf *tsdu;
uint8      protocol;
uint16     max0TSDUlen,max1TSDUlen;
uint8      version;
uint32     InitialSpsn;
uint8      TokenSetItem;
{
 uint8  PGI = 5;
 uint16 len1,len2,LI;
 len1 = tsdu->length;
 Build26PIU(tsdu,TokenSetItem);
 Build23PIU(tsdu,InitialSpsn);
 Build22PIU(tsdu,version);
 Build21PIU(tsdu,max0TSDUlen,max1TSDUlen);
 Build19PIU(tsdu,protocol);
 len2 = tsdu->length;
 if ((LI = len2 - len1) > ZERO)
    BuildHeader(tsdu,PGI,LI);
}

/*
 * Build User data
 */
static void Build193PGIU(tsdu)
struct buf *tsdu;
{
 uint8  PGI = 193;
 uint16 LI  = tsdu->length;
 if (LI > ZERO && LI <= 512)
    /* the (512 max) data bytes of User data are already in tsdu */
    BuildHeader(tsdu,PGI,LI);
}
```

```
/*
 * FUNCTION:  SPDU building functions
 *
 *            These functions are used to construct TSDUs from SPDUs.
 *            Given the appropriate parameters, each function prepends a
 *            particular SPDU onto a given TSDU buffer.
 *            To concatenate SPDUs onto a TSDU buffer, call the
 *            appropriate SPDU building functions in reverse order.
 *
 *            NOTE: The functions implemented here all build category 1
 *                  SPDUs, which are always mapped one-to-one onto a
 *                  TSDU. When building category 0 or 2 SPDUs, care
 *                  should be taken when calculating LI since the TSDU
 *                  may already contain other, concatenated SPDUs.
 *
 * INPUTS:    tsdu       - a pointer to the TSDU buffer. The .addr and
 *                         .length fields indicate the current TSDU size.
 *            parameters - the SPDU parameters.
 *
 * OUTPUTS:   The TSDU buffer .addr and .length fields are updated to
 *            include the prepended SPDU.
 *
 * CALLS:     BuildHeader, PIU and PGIU building functions
 */

/*
 * Build REFUSE SPDU - category 1
 */
static void BuildRF(tsdu,scid,
                    TCdis,
                    Srequirements,
                    version,
                    ReasonCode)
struct buf   *tsdu;
struct scid  *scid;
struct TCdis *TCdis;
uint16       Srequirements;
uint8        version;
uint8        ReasonCode;
{
 uint8  SI = 12;
 uint16 LI;
 Build50PIU(tsdu,ReasonCode);
 Build22PIU(tsdu,version);
 Build20PIU(tsdu,Srequirements);
 Build17PIU(tsdu,TCdis);
 Build1bPGIU(tsdu,scid);
 LI = tsdu->length;
 BuildHeader(tsdu,SI,LI);
}

/*
 * Build CONNECT SPDU - category 1
 */
static void BuildCN(tsdu,scid,
                    protocol,
                    max0TSDUlen,
                    max1TSDUlen,
                    version,
                    InitialSpsn,
                    TokenSetItem,
                    Srequirements,
                    clgSSAPid,
                    cldSSAPid)
```

```
struct buf     *tsdu;
struct scid    *scid;
uint8          protocol;
uint16         max0TSDUlen;
uint16         max1TSDUlen;
uint8          version;
uint32         InitialSpsn;
uint8          TokenSetItem;
uint16         Srequirements;
ssap_selector *clgSSAPid;
ssap_selector *cldSSAPid;
{
 uint8  SI = 13;
 uint16 LI;
 Build193PGIU(tsdu);
 Build52PIU(tsdu,cldSSAPid);
 Build51PIU(tsdu,clgSSAPid);
 Build20PIU(tsdu,Srequirements);
 Build5PGIU(tsdu,protocol,
                 max0TSDUlen,
                 max1TSDUlen,
                 version,
                 InitialSpsn,
                 TokenSetItem);
 Build1aPGIU(tsdu,scid);
 LI = tsdu->length;
 BuildHeader(tsdu,SI,LI);
}

/*
 * Build ACCEPT SPDU - category 1
 */
static void BuildAC(tsdu,scid,
                        protocol,
                        max0TSDUlen,
                        max1TSDUlen,
                        version,
                        InitialSpsn,
                        TokenSetItem,
                        TokenItem,
                        Srequirements,
                        clgSSAPid,
                        cldSSAPid)
struct buf     *tsdu;
struct scid    *scid;
uint8          protocol;
uint16         max0TSDUlen;
uint16         max1TSDUlen;
uint8          version;
uint32         InitialSpsn;
uint8          TokenSetItem;
uint16         Srequirements;
ssap_selector *clgSSAPid;
ssap_selector *cldSSAPid;
{
 uint8  SI = 14;
 uint16 LI;
 Build193PGIU(tsdu);
 Build52PIU(tsdu,cldSSAPid);
 Build51PIU(tsdu,clgSSAPid);
 Build20PIU(tsdu,Srequirements);
 Build16PIU(tsdu,TokenItem);
 Build5PGIU(tsdu,protocol,
                 max0TSDUlen,
                 max1TSDUlen,
                 version,
                 InitialSpsn,
                 TokenSetItem);
```

```
 Build1bPGIU(tsdu,scid);
 LI = tsdu->length;
 BuildHeader(tsdu,SI,LI);
}

/*
 * Build ABORT SPDU - category 1
 */
static void BuildAB(tsdu,TCdis,
                        ReflectParams)
struct buf    *tsdu;
struct TCdis  *TCdis;
struct Bytes9 *ReflectParams;
{
 uint8  SI = 25;
 uint16 LI;
 Build193PGIU(tsdu);
 Build49PIU(tsdu,ReflectParams);
 Build17PIU(tsdu,TCdis);
 LI = tsdu->length;
 BuildHeader(tsdu,SI,LI);
}

/*
 * Build ABORT ACCEPT SPDU - category 1
 */
static void BuildAA(tsdu)
struct buf *tsdu;
{
 uint8  SI = 26;
 uint16 LI = 0;
 BuildHeader(tsdu,SI,LI);
}

#undef ZERO
```

E.12 Miscellaneous functions source file listing: funcs2.c

```
/*
 * SESSION ENTITY STATIC FUNCTIONS: funcs2.c
 * ED van der Westhuizen   September 1989
 *
 * function          calls
 * --------          -----
 * reuseTC           StripHeader, Strip17PIU.
 * idSPDU            reuseTC.
 * UserAbort         balloc, BuildAB, TDTreq, SDTcnf, SpAc.
 * SessionError      printf.
 * get_buf           balloc, SessionError, os_exit.
 * get_mem           os_malloc, SessionError, os_exit.
 */

/*
 * FUNCTION:  reuseTC
 *
 *            This function determines whether a given RF, AB or FN SPDU
 *            requests re-use of the transport connection, i.e.:
 *            whether the SPDU is RFr,  ABr  or FNr  (TC reused),
 *                             or RFnr, ABnr or FNnr (TC not reused).
 *
 *            NOTE: RF, AB and FN are category 1 SPDUs, so only 1 SPDU
 *                  is expected in the given TSDU.
 *
 * INPUTS:    tsdu - a pointer to the TSDU containing the RF, AB or FN SPDU.
 *
 * OUTPUTS:   returns: TRUE  if SPDU is RFr,  ABr  or FNr,
 *                     FALSE if SPDU if RFnr, ABnr or FNnr.
 *
 * CALLS:     StripHeader, Strip17PIU.
 */

static boolean reuseTC(tsdu)
register struct buf *tsdu;
{
 uint8        *temp_addr  = tsdu->addr;
 int          temp_length = tsdu->length;
 uint16       LI;
 struct TCdis TCdis;
 StripHeader(tsdu,0);              /* remove the SPDU header */
 LI = StripHeader(tsdu,1);         /* strip PGIU_1 header if present */
 tsdu->addr   += LI;               /* skip PGIU_1 */
 tsdu->length -= LI;
 Strip17PIU(tsdu,&TCdis);          /* strip Transport disconnect parameter */
 tsdu->addr   = temp_addr;         /* restore tsdu parameters */
 tsdu->length = temp_length;
 return(TCdis.TCkept);
}
```

```
/*
 * FUNCTION:  idSPDU
 *
 *            This function determines whether the SPDU starting at the
 *            current position within a TSDU is the required SPDU.
 *
 * INPUTS:    tsdu   - a pointer to the TSDU buffer. The .addr and
 *                     .length fields indicate the current position.
 *            SPDUid - specifies the required SPDU.
 *
 * OUTPUTS:   returns: TRUE:  if the SPDU in the TSDU is the required one,
 *                     FALSE: if not.
 *
 * CALLS:     reuseTC.
 */

static boolean idSPDU(tsdu,SPDUid)
register struct buf *tsdu;
register int        SPDUid;
{
 if (tsdu->length) {
    uint8 SI = *(tsdu->addr);
    uint8 PI = *(tsdu->addr + 2);
    switch (SPDUid) {
    case RF  : return(SI == 12);
    case RFr : return(SI == 12 &&  reuseTC(tsdu));
    case RFnr: return(SI == 12 && !reuseTC(tsdu));
    case CN  : return(SI == 13);
    case AC  : return(SI == 14);
    case AB  : return(SI == 25 && PI == 17);
    case ABr : return(SI == 25 && PI == 17 &&  reuseTC(tsdu));
    case ABnr: return(SI == 25 && PI == 17 && !reuseTC(tsdu));
    case AA  : return(SI == 26);
    }
 }
 return(FALSE);   /* no more SPDUs in this TSDU */
}

/*
 * FUNCTION:  UserAbort
 *
 *            This function is called from session() when a SUABreq
 *            primitive is received. The TC may or may not be reused.
 *
 * INPUTS:    s      - the SCEP identifier.
 *            idu    - a pointer to the SIDU.
 *            TCkept - requests reuse or release of the TC:
 *                     TRUE  if TC is to be reused,
 *                     FALSE if not.
 *
 * OUTPUTS:   returns: TRUE:  if successful,
 *                     FALSE: if not.
 *
 * CALLS:     balloc, BuildAB, TDTreq, SDTcnf, SpAc.
 */

static boolean UserAbort(s,idu,TCkept)
register struct Smachine *s;
register struct idu      *idu;
register boolean         TCkept;
{
 if ((idu->buffer) || (idu->buffer = balloc(0))) {
    struct TCdis TCdis;
    TCdis.ABreason = USER_ABORT;
    TCdis.TCkept   = TCkept;          /* FALSE for ABnr, TRUE for ABr */
```

```
    BuildAB(idu->buffer,              /* TSDU with SSuser data */
            &TCdis,                   /* transport disconnect */
            NULL);                    /* reflect parameters */

    TDTreq(s->TCEPid,                 /* send AB */
           idu->buffer);
    SDTcnf(idu->buffer);              /* free the buffer */
    SpAc(4);                          /* start abort timer */
    return(TRUE);                     /* SUABreq successful */
  }
  return(FALSE);                      /* SUABreq failed */
}

/*
 * FUNCTION:  SessionError
 *
 *            This function prints session entity error messages.
 *
 * INPUTS:    errnum - identifies the error.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     printf.
 */

static void SessionError(errnum)
register int errnum;
{
 switch (errnum) {
 case SERBUF: printf("Session Error: buffer cannot be allocated.\n");
      break;
 case SERMEM: printf("Session Error: out of memory.\n");
      break;
 case SERSID: printf("Session Error: illegal SCEPid - S event ignored.\n");
      break;
 case SERTID: printf("Session Error: illegal TCEPid - T event ignored.\n");
      break;
 default:
      break;
 }
}

/*
 * FUNCTION:  get_buf
 *
 *            This function allocates a session layer buffer.
 *            If allocation fails, an error message is printed and
 *            the process exits.
 *
 * INPUTS:    size - the size of the buffer to allocate.
 *
 * OUTPUTS:   returns: a pointer to the allocated buffer.
 *
 * CALLS:     balloc, SessionError, os_exit.
 */

static struct buf *get_buf(size)
register int size;
{
 struct buf *buf;
 if ((buf = balloc(size)) == NULL) {
    SessionError(SERBUF);
    os_exit(0);                       /* no buffers, so exit */
 }
 return(buf);
}
```

```
/*
 * FUNCTION:  get_mem
 *
 *            This function allocates memory. If allocation fails,
 *            an error message is printed and the process exits.
 *
 * INPUTS:    size - the number of bytes to be allocated.
 *
 * OUTPUTS:   returns: a pointer to the first byte of the allocated
 *                     memory area.
 *
 * CALLS:     os_malloc, SessionError, os_exit.
 */

static char *get_mem(size)
register unsigned size;
{
 char *mem;
 if ((mem = os_malloc(size)) == NULL) {
    SessionError(SERMEM);
    os_exit(0);                    /* no memory, so exit */
 }
 return(mem);
}
```

E.13 Session entity archive makefile listing: makefile

```
# MAKEFILE FOR SESSION ENTITY ARCHIVE (session.a): makefile
# ED van der Westhuizen   September 1989
#
# pre-processor flags:
# -DLINT_ARGS  include argument types in function declarations (not for lint)
# -DDEBUG      include session debug code

.SUFFIXES:              # clear all suffixes
.SUFFIXES: .o .ln .c .h  # new suffixes
INCLDS = -I../include    # also search here for #include "" files

# cc compiler:          flags: optimise
CFLAGS = $(INCLDS) -DDEBUG -O

# lint program checker:  flags: no-heuristic no-unused ln-only
LFLAGS = $(INCLDS) -DDEBUG -hvc

# ar archive maintainer: flags: replace updated verbose
AFLAGS = -ruv

# pathname to Existing X.400 Product header files:
P      = ../include/

OBJS   = session.o buffer.o       # archive component object files
LINTS  = session.ln buffer.ln     # lint 1st pass files (externals)

# session.c #include files:
SINCLD = debug.c sprmtvs.c tprmtvs.c funcs1.c build.c strip.c funcs2.c \
         sconfig.h buffer.h trans.h \
         $Posdeps.h $Psystem.h $Ptransport.h $Paccess.h $Paddress.h \
         $Psession.h $Pqueue.h

# buffer.c #include files:
BINCLD = $Posdeps.h $Paddress.h $Psession.h $Pqueue.h $Ptransport.h

#
#  AR:  make session.a   or   make
#

# update the archive:
session.a: $(OBJS)  ;ar $(AFLAGS) session.a $(OBJS)

# object file dependencies besides the .c default:
session.o: $(SINCLD)
buffer.o:  $(BINCLD)

#
#  LINT:  make lint
#
#  lint 1st pass external info in .ln files
#  intra-file bugs in .er files
#

# user-defined suffix-rule .c.ln:
.c.ln:  ;lint $(LFLAGS) $*.c > $*.er

# linting the archive files - lint 1st pass only
lint: $(LINTS)

# .ln file dependencies besides the .c default:
session.ln: $(SINCLD)
buffer.ln:  $(BINCLD)
```

F.1 Transport entity source file listing: transport.c

```
/*
 * TRANSPORT ENTITY SOURCE FILE: transport.c
 * ED van der Westhuizen   September 1989
 *
 * This code provides a simple, pseudo Transport Layer between two
 * correspondent RTS/SESSION processes. One TSAP is provided to each
 * process, with one transport connection between them.
 *
 * External visibility:
 *
 *    void    TSUadd()
 *    pointer UCONreq()
 *    void    UCONres()
 *    void    UDISreq()
 *    void    UDATreq()
 */

/*-------------------------------------------------------------*
 * HEADER FILES                                                *
 *-------------------------------------------------------------*/

#include "osdeps.h"     /* os dependent definitions and macros */
#include "address.h"    /* address definitions */
#include "transport.h"  /* transport interface definitions */
#include "session.h"    /* session interface definitions */

#include <sys/ipc.h>    /* inter-process communication */
#include <sys/msg.h>    /* ipc: messages */
#include <errno.h>      /* error messages */

/*-------------------------------------------------------------*
 * MANIFEST CONSTANTS                                          *
 *-------------------------------------------------------------*/

#define MAXTSDULEN  800          /* maximun TSDU length */
                                 /* NB: this value is purely for testing
                                  * purposes and is determined by the
                                  * ipc message queue size. It is not related
                                  * to the value used by the session protocol.
                                  */
/*
 * inter-process message queue control
 */

#define KEY      2               /* message queue key */
#define USR_R_W  (00400 + 00200) /* user read/write permission */

/*
 * transport request/response primitive identifiers
 */

#define TCONREQ  4  /* T-CONNECT.request */
#define TCONRSP  5  /* T-CONNECT.response */
#define TDTREQ   6  /* T-DATA.request */
#define TDISREQ  7  /* T-DISCONNECT.request */
```

```
/*-------------------------------------------------------*
 * DATA TYPES AND STRUCTURES                             *
 *-------------------------------------------------------*/

/*
 * Transport Interface Data Unit. This structure carries transport
 * primitive data between the two RTS/SESSION/TRANSPORT processes.
 * It is mapped into a message queue buffer for transfer.
 *
 * NOTE: TIDU size = 5332 for session layer buffer size of 5130 bytes.
 *       However, max message queue size only = 4096 bytes.
 */

typedef struct {
           pointer       TCEPid;              /* TCEP identifier */
           uint8         event;               /* T event identifier */
           nsap_address  clgNSAPaddr,         /* calling NSAP address */
                         cldNSAPaddr;         /* called  NSAP address */
           tsap_selector clgTSAPid,           /* calling TSAPid */
                         cldTSAPid;           /* called  TSAPid */
           qos_type      qots;                /* QOTS values */
           boolean       texp;                /* T-EXPEDITED-DATA option */
           uint8         reason;              /* disconnect reason */
           pointer       TSUdata;             /* TS-user data */
           boolean       eotsdu;              /* End Of TSDU flag */
           int           length;              /* TSDU length */
           uint8         buffer[MAXTSDULEN];  /* TSDU buffer */
       } TIDU;

/*
 * message queue buffer
 */

struct msgbuf1 {
           long mtype;
           char mtext[sizeof(TIDU)];
};

/*-------------------------------------------------------*
 * STATIC VARIABLE DEFINITIONS                           *
 *-------------------------------------------------------*/

/*
 * local NSAP address
 */

static nsap_address localNSAPaddr = {
                              3,
                              0xA1,0xA1,0xA1
                   };

static int (*TCONind)();  /* TS-user TCONind handler */
static int (*TCONcnf)();  /* TS-user TCONcnf handler */
static int (*TDISind)();  /* TS-user TDISind handler */
static int (*TDTind)();   /* TS-user TDTind  handler */

static long in_type;      /* input  message type */
static long out_type;     /* output message type */
static int  msqid;        /* message queue identifier */
```

```
/*-------------------------------------------------------*
 * STATIC FUNCTION DEFINITIONS                           *
 *-------------------------------------------------------*/

/*
 * FUNCTION:  GracefulExit
 *
 *            This function is called when the process catches a 'kill'
 *            signal. It brings down the inter-process message queue
 *            and exits.
 *
 * INPUTS:    none.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     msgctl, printf, os_exit.
 */

void GracefulExit()
{
 if (msgctl(msqid, IPC_RMID, NULL) == -1)
    printf("\nmsgctl failed. errno = %d\n",errno);
 printf("\n\nRTS terminated by kill.\n");
 os_exit(0);
}

/*-------------------------------------------------------*
 * EXTERNAL FUNCTION DEFINITIONS                         *
 *-------------------------------------------------------*/

/*
 * FUNCTION:  TSUadd
 *
 *            The TS-user calls this function to register itself with
 *            the transport entity, and to initialize the transport
 *            entity. This function also sets up the inter-process
 *            message queue.
 *
 * INPUTS:    tsap_id - pointer to local TSAPid.
 *            tconind - pointer to TS-user TCONind handler function.
 *            tconcnf - pointer to TS-user TCONcnf handler function.
 *            tdisind - pointer to TS-user TDISind handler function.
 *            tdtind  - pointer to TS-user TDTind  handler function.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     os_sigset, msgget, printf, os_exit.
 */

void TSUadd(tsap_id,tconind,tconcnf,tdisind,tdtind)
tsap_selector *tsap_id;
int           (*tconind)();
int           (*tconcnf)();
int           (*tdisind)();
int           (*tdtind)();
{
 TCONind = tconind;
 TCONcnf = tconcnf;
 TDISind = tdisind;
 TDTind  = tdtind;

 os_sigset(SIGTERM, GracefulExit);  /* catch SIGTERM for graceful exit */
```

```
    /*
     * Set up the inter-process message queue
     */

    if ((msqid = msgget(KEY, (IPC_CREAT | IPC_EXCL | USR_R_W))) == -1) {
       if ((msqid = msgget(KEY, (IPC_CREAT | USR_R_W))) == -1) {
          printf("\nmsgget failed. errno = %d\n",errno);
          os_exit(0);
       }
       else {              /* the other process is the creator */
          in_type  = 1;
          out_type = 2;
       }
    }
    else {                 /* this process is the creator */
       in_type  = 2;
       out_type = 1;
    }
}

/*
 * FUNCTIONs:  Transport request/response primitive handlers
 *
 *             These functions each receive a particular T req/rsp primitive
 *             from the TS-user and pass it to the correspondent transport
 *             entity via the inter-process message queue. Each function maps
 *             all its primitive data into a message queue output buffer
 *             using the TIDU structure as a template. This buffer is then
 *             placed on the message queue.
 *
 * INPUTS:     see individual functions.
 *
 * OUTPUTS:    All these functions return void, except UCONreq. It returns
 *             a dummy TCEPid to the TS-user.
 *
 * CALLS:      msgsnd, bcopy, printf.
 */

/*
 *  primitive:  T-CONNECT.request
 */

pointer UCONreq(clgTSAPid,cldNSAPaddr,cldTSAPid,qots,texp,TSUdata)
tsap_selector *clgTSAPid;   /* calling TSAP id */
nsap_address  *cldNSAPaddr; /* called  NSAP address */
tsap_selector *cldTSAPid;   /* called  TSAP id */
qos_type      *qots;        /* proposed QOTS */
boolean       texp;         /* proposed TEXP option */
uint8         *TSUdata;     /* TS-user data */
{
  struct msgbuf1 msgbuf;
  TIDU           *tidu = (TIDU *) msgbuf.mtext;

  msgbuf.mtype  = out_type;
  tidu->event   = TCONREQ;
  tidu->TCEPid  = (pointer) 2;    /* "allocate" a TCEPid */
  tidu->texp    = texp;
  tidu->TSUdata = TSUdata;
  bcopy(clgTSAPid,      &tidu->clgTSAPid,   sizeof(tsap_selector));
  bcopy(cldNSAPaddr,    &tidu->cldNSAPaddr, sizeof(nsap_address));
  bcopy(cldTSAPid,      &tidu->cldTSAPid,   sizeof(tsap_selector));
  bcopy(&localNSAPaddr, &tidu->clgNSAPaddr, sizeof(nsap_address));
  bcopy(qots,           &tidu->qots,        sizeof(qos_type));

  if (msgsnd(msqid, &msgbuf, sizeof(TIDU), IPC_NOWAIT) == -1)
     printf("\nmsgsnd failed. errno = %d\n",errno);

  return((pointer) 2);            /* "allocated" TCEPid */
}
```

```
/*
 *  primitive:  T-CONNECT.response
 */

void UCONres(TCEPid,qots,texp,TSUdata)
pointer  TCEPid;   /* TCEPid */
qos_type *qots;    /* selected QOTS */
boolean  texp;     /* selected TEXP option */
uint8    *TSUdata; /* TS-user data */
{
  struct msgbuf1 msgbuf;
  TIDU           *tidu = (TIDU *) msgbuf.mtext;

  msgbuf.mtype  = out_type;
  tidu->event   = TCONRSP;
  tidu->TCEPid  = TCEPid;
  tidu->texp    = texp;
  tidu->TSUdata = TSUdata;
  bcopy(qots, &tidu->qots, sizeof(qos_type));

  if (msgsnd(msqid, &msgbuf, sizeof(TIDU), IPC_NOWAIT) == -1)
     printf("\nmsgsnd failed. errno = %d\n",errno);
}

/*
 *  primitive:  T-DISCONNECT.request
 */

void UDISreq(TCEPid,TSUdata)
pointer TCEPid;   /* TCEPid */
uint8   *TSUdata; /* TS-user data */
{
  struct msgbuf1 msgbuf;
  TIDU           *tidu = (TIDU *) msgbuf.mtext;

  msgbuf.mtype  = out_type;
  tidu->event   = TDISREQ;
  tidu->TCEPid  = TCEPid;
  tidu->TSUdata = TSUdata;

  if (msgsnd(msqid, &msgbuf, sizeof(TIDU), IPC_NOWAIT) == -1)
     printf("\nmsgsnd failed. errno = %d\n",errno);
}

/*
 *  primitive:  T-DATA.request
 */

void UDATreq(TCEPid,tsdu,eotsdu)
pointer    TCEPid; /* TCEPid */
struct buf *tsdu;  /* TSDU buffer */
boolean    eotsdu; /* end of TSDU flag */
{
  struct msgbuf1 msgbuf;
  TIDU           *tidu = (TIDU *) msgbuf.mtext;

  msgbuf.mtype = out_type;
  tidu->event  = TDTREQ;
  tidu->TCEPid = TCEPid;
  tidu->eotsdu = eotsdu;
  tidu->length = tsdu->length;
  bcopy(tsdu->addr, tidu->buffer, tsdu->length);

  if (msgsnd(msqid, &msgbuf, sizeof(TIDU), IPC_NOWAIT) == -1)
     printf("\nmsgsnd failed. errno = %d\n",errno);
}
```

```
/*
 * FUNCTION:  do_transport-queue
 *
 *            The TS-user calls this function to process the incoming
 *            T req/rsp primitives in the message queue. T req/rsp primitives
 *            are converted to T ind/cnf primitives and the appropriate TS-user
 *            T ind/cnf handlers are called. Each incoming primitive in the
 *            queue is processed until the queue is empty. The TS-user
 *            T ind/cnf handlers were passed to the transport entity in
 *            a previous call to TSUadd().
 *
 * INPUTS:    none.
 *
 * OUTPUTS:   none.
 *
 * CALLS:     TS-user T ind/cnf handlers:
 *            (*TCONind)(), (*TCONcnf)(), (*TDTind)(), (*TDISind)(),
 *            msgrcv, printf, os_exit.
 */

void do_transport_queue()
{
 struct msgbuf1 msgbuf;
 TIDU          *tidu = (TIDU *) msgbuf.mtext;

 while (msgrcv(msqid, &msgbuf, sizeof(TIDU), in_type, IPC_NOWAIT) != -1) {
       switch (tidu->event) {
       case TCONREQ:                        /* T-CONNECT.request */
            (*TCONind)(tidu->TCEPid,        /* send T-CONNECT.indication */
                       &tidu->clgNSAPaddr,
                       &tidu->clgTSAPid,
                       &tidu->cldTSAPid,
                       &tidu->qots,
                       tidu->texp,
                       tidu->TSUdata);
            break;

       case TCONRSP:                        /* T-CONNECT.response */
            (*TCONcnf)(tidu->TCEPid,        /* send T-CONNECT.confirm */
                       &tidu->cldNSAPaddr,
                       &tidu->cldTSAPid,
                       &tidu->qots,
                       tidu->texp,
                       tidu->TSUdata);
            break;

       case TDISREQ:                        /* T-DISCONNECT.request */
            (*TDISind)(tidu->TCEPid,        /* send T-DISCONNECT.indication */
                       0,
                       tidu->TSUdata);
            break;

       case TDTREQ:                         /* T-DATA.request */
            {
             struct buf tsdu;
             tsdu.addr   = tidu->buffer;
             tsdu.length = tidu->length;
             (*TDTind)(tidu->TCEPid,        /* send T=DATA.indication */
                       &tsdu,
                       tidu->eotsdu);
             break;
            };
       }
 }

 switch (errno) {    /* error handling */
 case ENOMSG:        /* no input messages - as expected */
      break;
```

```
    default:            /* any other error */
        printf("\n\nThe message queue is down, status = %d. ",errno);
        printf("RTS terminated.\n");
        os_exit(0);
        break;
    }
}
```

F.2 Transport entity object file makefile listing: makefile

```
# MAKEFILE FOR TRANSPORT ENTITY OBJECT CODE (transport.o): makefile
# ED van der Westhuizen   September 1989
#
# pre-processor flags:
# -DLINT_ARGS  include argument types in function declarations (not for lint)

.SUFFIXES:               # clear all suffixes
.SUFFIXES: .o .ln .c .h  # new suffixes
INCLDS = -I../include    # also search here for #include "" files

# cc compiler:           flags: optimise
CFLAGS = $(INCLDS) -DLINT_ARGS -O

# lint program checker:  flags: no-heuristic no-unused ln-only
LFLAGS = $(INCLDS) -DLINT_ARGS -hvc

# pathname to Existing X.400 Product header files:
P       = ../include/

# transport.c #include files:
TINCLD = $Posdeps.h $Ptransport.h $Paddress.h

#
# COMPILE: make or make transport
#

transport: transport.o

# .o file dependencies besides the .c default:
transport.o: $(TINCLD)

#
#  LINT:  make lint
#
#  lint 1st pass external info in .ln file
#  intra-file bugs in .er file
#

# user-defined suffix-rule .c.ln:
.c.ln:  ;lint $(LFLAGS) $*.c > $*.er

# linting the transport file - lint 1st pass only
lint: transport.ln

# .ln file dependencies besides the .c default:
transport.ln: $(TINCLD)
```

APPENDIX G. Test Outputs

This appendix lists the session entity debug outputs for the two software tests:

G.1 Test 1: Successful Session Connection Establishment.
G.2 Test 2: Unsuccessful Session Connection Establishment.

For each test, the actions of the two session entities are numbered sequentially to show their interleaving. This numbering is of the following format:

**** N ****

G.1 Test 1: Successful Session Connection Establishment

***** Debug output of Session Entity 1. *****

***** RTS 1 is the calling SS-user. *****

```
** 1 **
init_session(): Initialize the session entity.
   local TSAPid: B1 B1 B1

** 1 **
s_activate(): Register a SS-user.
   local SSAPid: C1 C1 C1
   return TRUE

** 2 **
s_connect(): Allocate a SPM for a session connection.
   local  SSAPid:   C1 C1 C1
   remote NSAPaddr: A2 A2 A2   remote TSAPid:   B2 B2 B2
   remote SSAPid:   C2 C2 C2
   SPM(0) with free TC allocated.

** 3 **
session(): Session req/rsp primitive received.
   Input event:  S-CONNECT.request
   SPM:          0
   From state:   STA01 - idle, no TC
   Output event: T-CONNECT.request (successful)
   To state:     STA01B - await T-CONNECT.confirm
   return TRUE

** 5 **
do_session_queue(): Transport ind/cnf primitive received.
   Input event:  T-CONNECT.confirm
   SPM:          0
   From state:   STA01B - await T-CONNECT.confirm
   Output event: T-DATA.request

   The SPDU is CONNECT:
   013 094 001 022 010 003 049 049 049 011 015 023 013 057 048 048 049 050 052
   049 057 048 054 049 056 090 005 015 019 001 000 021 004 004 000 004 000 022
   001 001 026 001 000 020 002 002 073 051 003 193 193 193 052 003 194 194 194
   193 037 049 128 160 128 128 001 000 000 000 161 128 128 001 005 129 001 006
   130 001 000 163 128 160 128 005 000 000 000 000 000 132 001 001 000 000 000
   000

   Output event: S-DATA.confirm
   To state:     STA02A - await ACCEPT SPDU

** 8 **
do_session_queue(): Transport ind/cnf primitive received.
   Input event:  T-DATA.indication

   The SPDU is ACCEPT:
   014 088 001 022 009 003 049 049 049 011 015 023 013 057 048 048 049 050 052
   049 057 048 054 049 056 090 005 015 019 001 000 021 004 004 000 004 000 022
   001 001 026 001 000 020 002 002 073 051 003 193 193 193 052 003 194 194 194
   193 031 049 128 160 128 128 001 000 000 000 161 128 128 001 005 129 001 006
   162 128 160 128 005 000 000 000 000 000 000 000 000 000 000
```

```
   SPM:          0
   From state:   STA02A - await ACCEPT SPDU

   SPDU parameters:
   scid.cld_ref = 31 31 31
   scid.com_ref = 17 0D 39 30 30 31 32 34 31 39 30 36 31 38 5A
   scid.add_ref =
   prtcl opns   = 0
   max0TSDUlen  = 1024
   max1TSDUlen  = 1024
   version      = 1
   token        = 0
   tokenItem    = 0
   initial sn   = 0
   fus          = 585
   clgSSAPid    = C1 C1 C1
   cldSSAPid    = C2 C2 C2
   SS-user data:
   049 128 160 128 128 001 000 000 000 161 128 128 001 005 129 001 006 162 128
   160 128 005 000 000 000 000 000 000 000 000 000

   Output event: S-CONNECT.confirm (accept)
   To state:     STA713 - data transfer

** 9 **
session(): Session req/rsp primitive received.
   Input event:  S-ACTIVITY-START.request
   SPM:          0
   From state:   STA713 - data transfer
   To state:     STA713 - data transfer
   return TRUE

** 10 **
session(): Session req/rsp primitive received.
   Input event:  S-DATA.request
   SPM:          0
   From state:   STA713 - data transfer
   To state:     STA713 - data transfer
   return TRUE

** 11 **
session(): Session req/rsp primitive received.
   Input event:  S-ACTIVITY-END.request
   SPM:          0
   From state:   STA713 - data transfer
   Output event: T-DISCONNECT.request
   To state:     STA713 - data transfer
   return TRUE

** 12 **
do_session_queue(): Transport ind/cnf primitive received.
   Input event:  T-DISCONNECT.indication
   SPM:          0
   From state:   STA713 - data transfer
   Output event: S-PROVIDER-ABORT.indication
   To state:     STA01 - idle, no TC

The message queue is down, status = 22. RTS terminated.
```

```
*** Debug output of Session Entity 2.  ***
*** RTS 2 is the called SS-user.       ***


** 1 **
init_session(): Initialize the session entity.
   local TSAPid: B2 B2 B2


** 1 **
s_activate(): Register a SS-user.
   local SSAPid: C2 C2 C2
   return TRUE


** 4 **
do_session_queue(): Transport ind/cnf primitive received.
   Input event:  T-CONNECT.indication
         remote NSAPaddr: A1 A1 A1
         remote TSAPid:   B1 B1 B1
   SPM:          0
   From state:   STA01 - idle, no TC
   Output event: T-CONNECT.response
   To state:     STA01C - idle, TC connected


** 6 **
do_session_queue(): Transport ind/cnf primitive received.
   Input event:  T-DATA.indication

   The SPDU is CONNECT:
   013 094 001 022 010 003 049 049 049 011 015 023 013 057 048 048 049 050 052
   049 057 048 054 049 056 090 005 015 019 001 000 021 004 004 000 004 000 022
   001 001 026 001 000 020 002 002 073 051 003 193 193 193 052 003 194 194 194
   193 037 049 128 160 128 128 001 000 000 000 161 128 128 001 005 129 001 006
   130 001 000 163 128 160 128 005 000 000 000 000 000 132 001 001 000 000 000
   000

   SPM:          0
   From state:   STA01C - idle, TC connected

   SPDU parameters:
   scid.clg_ref = 31 31 31
   scid.com_ref = 17 0D 39 30 30 31 32 34 31 39 30 36 31 38 5A
   scid.add_ref =
   protocol     = 0
   max0TSDUlen  = 1024
   max1TSDUlen  = 1024
   version      = 1
   token        = 0
   initial sn   = 0
   fus          = 585
   clgSSAPid    = C1 C1 C1
   cldSSAPid    = C2 C2 C2
   SS-user data:
   049 128 160 128 128 001 000 000 000 161 128 128 001 005 129 001 006 130 001
   000 163 128 160 128 005 000 000 000 000 000 132 001 001 000 000 000 000

   Output event: S-CONNECT.indication
         remote NSAPaddr: A1 A1 A1
         remote TSAPid:   B1 B1 B1
         remote SSAPid:   C1 C1 C1
   To state:     STA08 - await S-CONNECT.response
```

```
** 7 **
session(): Session req/rsp primitive received.
   Input event:  S-CONNECT.response (accept)
   SPM:          0
   From state:   STA08 - await S-CONNECT.response
   Output event: T-DATA.request

   The SPDU is ACCEPT:
   014 088 001 022 009 003 049 049 049 011 015 023 013 057 048 048 049 050 052
   049 057 048 054 049 056 090 005 015 019 001 000 021 004 004 000 004 000 022
   001 001 026 001 000 020 002 002 073 051 003 193 193 193 052 003 194 194 194
   193 031 049 128 160 128 128 001 000 000 000 161 128 128 001 005 129 001 006
   162 128 160 128 005 000 000 000 000 000 000 000 000 000

   To state:     STA713 - data transfer
   return TRUE


** 12 **
do_session_queue(): Transport ind/cnf primitive received.
   Input event:  T-DISCONNECT.indication
   SPM:          0
   From state:   STA713 - data transfer
   Output event: S-PROVIDER-ABORT.indication
   To state:     STA01 - idle, no TC


RTS terminated by kill.
```

G.2 Test 2: Unsuccessful Session Connection Establishment

***** Debug output of Session Entity 1. *****

***** RTS 1 is the calling SS-user. *****

```
** 1 **
init_session(): Initialize the session entity.
   local TSAPid: B1 B1 B1

** 1 **
s_activate(): Register a SS-user.
   local SSAPid: C1 C1 C1
   return TRUE

** 2 **
s_connect(): Allocate a SPM for a session connection.
   local  SSAPid:   C1 C1 C1
   remote NSAPaddr: A2 A2 A2
   remote TSAPid:   B2 B2 B2
   remote SSAPid:   C2 C2 C2
   SPM(0) with free TC allocated.

** 3 **
session(): Session req/rsp primitive received.
   Input event:  S-CONNECT.request
   SPM:          0
   From state:   STA01 - idle, no TC
   Output event: T-CONNECT.request (successful)
   To state:     STA01B - await T-CONNECT.confirm
   return TRUE

** 5 **
do_session_queue(): Transport ind/cnf primitive received.
   Input event:  T-CONNECT.confirm
   SPM:          0
   From state:   STA01B - await T-CONNECT.confirm
   Output event: T-DATA.request

   The SPDU is CONNECT:
   013 094 001 022 010 003 049 049 049 011 015 023 013 057 048 048 049 050 052
   049 057 050 054 049 056 090 005 015 019 001 000 021 004 004 000 004 000 022
   001 001 026 001 000 020 002 002 073 051 003 193 193 193 052 003 194 194 194
   193 037 049 128 160 128 128 001 000 000 000 161 128 128 001 005 129 001 006
   130 001 000 163 128 160 128 005 000 000 000 000 000 132 001 001 000 000 000
   000

   Output event: S-DATA.confirm
   To state:     STA02A - await ACCEPT SPDU
```

```
** 8 **
do_session_queue(): Transport ind/cnf primitive received.
   Input event:  T-DATA.indication

   The SPDU is ABORT (reuse):
   025 012 017 001 002 193 007 049 128 128 001 004 000 000

   SPM:          0
   From state:   STA02A - await ACCEPT SPDU
   Output event: S-USER-ABORT.indication
   Output event: T-DATA.request

   The SPDU is ABORT ACCEPT:
   026 000

   To state:     STA01C - idle, TC connected

** 10 **
SessionTimeOut(): A timer has expired.
   SPM:          0
   From state:   STA01C - idle, TC connected
   Input event:  Connect timer timeout
   Output event: T-DISCONNECT.request
   To state:     STA01 - idle, no TC

** 13 **
do_session_queue(): Transport ind/cnf primitive received.
   Input event:  T-DISCONNECT.indication
   SPM:          0
   From state:   STA01 - idle, no TC
   To state:     STA01 - idle, no TC


The message queue is down, status = 22. RTS terminated.
```

```
*** Debug output of Session Entity 2.  ***
*** RTS 2 is the called SS-user.       ***


** 1 **
init_session(): Initialize the session entity.
   local TSAPid: B2 B2 B2

** 1 **
s_activate(): Register a SS-user.
   local SSAPid: C2 C2 C2
   return TRUE

** 4 **
do_session_queue(): Transport ind/cnf primitive received.
   Input event:  T-CONNECT.indication
         remote NSAPaddr: A1 A1 A1
         remote TSAPid:   B1 B1 B1
   SPM:          0
   From state:   STA01 - idle, no TC
   Output event: T-CONNECT.response
   To state:     STA01C - idle, TC connected

** 6 **
do_session_queue(): Transport ind/cnf primitive received.
   Input event:  T-DATA.indication

   The SPDU is CONNECT:
   013 094 001 022 010 003 049 049 049 011 015 023 013 057 048 048 049 050 052
   049 057 050 054 049 056 090 005 015 019 001 000 021 004 004 000 004 000 022
   001 001 026 001 000 020 002 002 073 051 003 193 193 193 052 003 194 194 194
   193 037 049 128 160 128 128 001 000 000 000 161 128 128 001 005 129 001 006
   130 001 000 163 128 160 128 005 000 000 000 000 000 132 001 001 000 000 000
   000

   SPM:          0
   From state:   STA01C - idle, TC connected

   SPDU parameters:
   scid.clg_ref = 31 31 31
   scid.com_ref = 17 0D 39 30 30 31 32 34 31 39 32 36 31 38 5A
   scid.add_ref =
   protocol     = 0
   max0TSDUlen  = 1024
   max1TSDUlen  = 1024
   version      = 1
   token        = 0
   initial sn   = 0
   fus          = 585
   clgSSAPid    = C1 C1 C1
   cldSSAPid    = C2 C2 C2
   SS-user data:
   049 128 160 128 128 001 000 000 000 161 128 128 001 005 129 001 006 130 001
   000 163 128 160 128 005 000 000 000 000 000 132 001 001 000 000 000 000

   Output event: S-CONNECT.indication
         remote NSAPaddr: A1 A1 A1
         remote TSAPid:   B1 B1 B1
         remote SSAPid:   C1 C1 C1
   To state:     STA08 - await S-CONNECT.response
```

```
** 7 **
session(): Session req/rsp primitive received.
   Input event:  S-USER-ABORT.request
   SPM:          0
   From state:   STA08 - await S-CONNECT.response
   Output event: T-DATA.request

   The SPDU is ABORT (reuse):
   025 012 017 001 002 193 007 049 128 128 001 004 000 000

   Output event: S-DATA.confirm
   To state:     STA01A - await ABORT ACCEPT SPDU
   return TRUE


** 9 **
do_session_queue(): Transport ind/cnf primitive received.
   Input event:  T-DATA.indication

   The SPDU is ABORT ACCEPT:
   026 000

   SPM:          0
   From state:   STA01A - await ABORT ACCEPT SPDU
   To state:     STA01C - idle, TC connected


** 11 **
SessionTimeOut(): A timer has expired.
   SPM:          0
   From state:   STA01C - idle, TC connected
   Input event:  Connect timer timeout
   Output event: T-DISCONNECT.request
   To state:     STA01 - idle, no TC


** 12 **
do_session_queue(): Transport ind/cnf primitive received.
   Input event:  T-DISCONNECT.indication
   SPM:          0
   From state:   STA01 - idle, no TC
   To state:     STA01 - idle, no TC


RTS terminated by kill.
```