INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road. Ann Arbor. MI 48106-1346 USA
313/761-4700 800/521-0600

Specification and Verification of the Service Specific Connection Oriented Protocol

MORTEZA GHODRAT

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

APRIL 1995

Abstract

Specification and Verification of the Service Specific Connection Oriented Protocol

Morteza Ghodrat

To permit negotiation of connections over the Asynchronous Transfer Mode, a signaling function is required. This signaling function, in turn, makes use of an assured data delivery service. This service is provided by the Service Specific Connection Oriented Protocol (SSCOP), which resides in the ATM Adaptation Layer (AAL).

A specification of the SSCOP has been developed in Estelle. In addition, testing modules representing the action of the user, the action of the management entity and the fault module of the underlying unassured data delivery service have been developed.

The testing modules, used in conjunction with the "observers" feature of the Estelle Development Tooset (EDT), provide a framework for the verification of the protocol. Using a series of scenarios, it has been shown that the protocol and its service interface meet the functional specifications outlined in the Protocol Definition document.

Acknowledgments

I would like to take this opportunity to express my sincerest thanks to my supervisor, Dr. J.W. Atwood, for his advice, encouragement, and practical assistance during the course of this research.

I also wish to thank Mr. Dominique Bonjour of the Centre National D'Études des Télécommunications in Lannion, France, for providing the original motivation for this study, and for making available an Estelle specification of an earlier version of the SSCOP.

The financial support of the Fonds pour la Formation de Chercheurs et l'Aide à la Recherche through a grant to Dr. Atwood is also gratefully acknowledged.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Signaling Functions in the ATM Adaptation Layer

In the emerging field of high speed networking, Asynchronous Transfer Mode (ATM) is a key component. ATM is a telecommunication concept defined by ANSI and ITU-T (formerly CCITT) standards for carriage of a complete range of traffic, including voice, data, video, etc. As such, ATM technology can be used to aggregate user traffic from existing applications (private lines, video conference circuits, ...) to a single network interface, and to facilitate multi-media networking between high speed devices. The ATM bearer service[1] [2] as defined by ANSI and ITU-T [3] [4] standards provides a sequence-preserving, connection-oriented

[1] Also referred to as the B-ISDN, class X service

cell transfer between source and destination with agreed Quality of Service (QoS) and throughput.

As outlined in ITU-T Draft Recommendation Q.2110 [1], the ATM Adaptation Layer (AAL) is defined as enhancing the services provided by the ATM layer to support the functions required by the next higher layer. Different AALs support various protocols to suit the different needs of a range of AAL service users. One of these, AAL type 5 [8], is used for delivery of asynchronous data.

One particular type of AAL service user is the signaling entity wishing to communicate with its peer entity. The signaling AAL (SAAL) provides the functionality necessary to support such a signaling entity. The SAAL is divided into two sublayers, as shown in the figure 1.1, each of which is further divided into two parts. The Common Part encompasses functions that are common to all services in the AAL type 5. It is composed of the Common

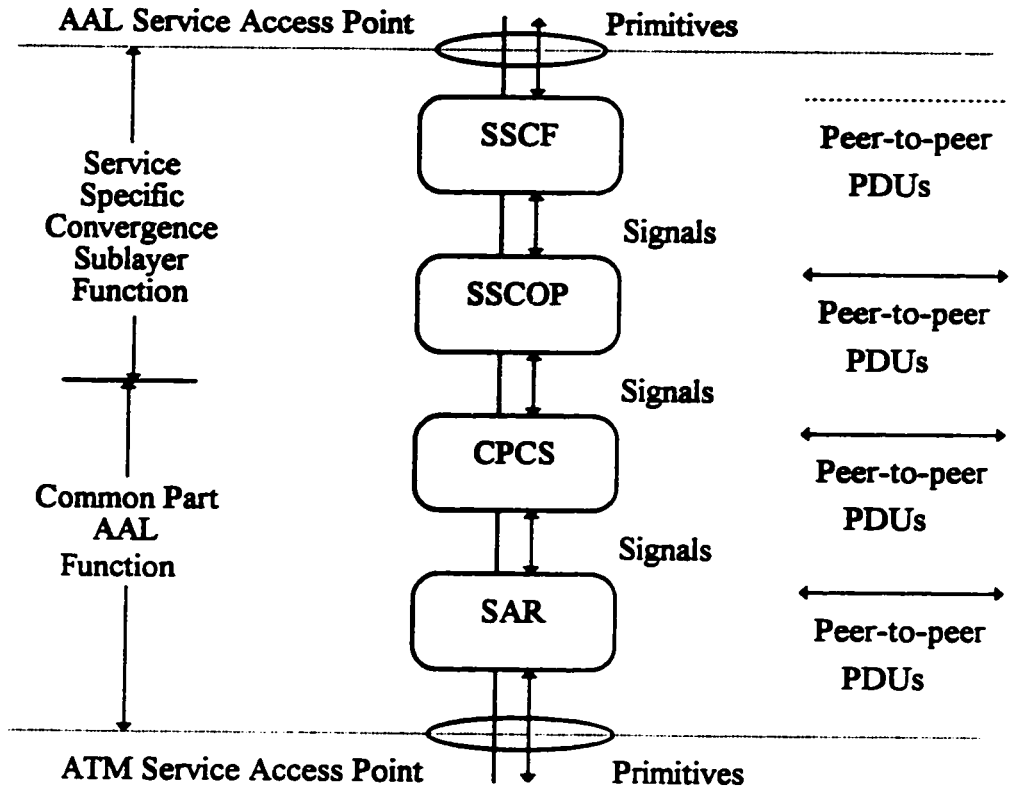


Figure 1.1: Signaling AAL structure

Two SSCFs that are currently defined are the SSCF-UNI (Q.2130) [11], which maps the particular requirements of the User-to-Network Interface protocol to the SSCOP services, and the SSCF-NNI (Q.2140) [12], which maps the particular requirements of the Network-to-Network Interface protocol to the SSCOP services.

The SSCOP provides mechanisms for the establishment and release of connections, and the reliable exchange of information between peer entities. Specification and validation of the SSCOP are the goals of this thesis.

The Common Part Convergence Sublayer provides transparent transport of the Service Data Units (SDUs) produced by the next higher layer (the SSCOP). The Segmentation and Reassembly function segments the SDUs to fit into outgoing ATM cells, and reassembles the incoming ATM cells.

1.2 Scope of the Thesis

In this paper our work splits into two parts:

1. To specify the SSCOP (presented in Recommendation Q.2110 , in the form of SDL diagrams [1]) in Estelle. Estelle is an ISO standard for protocol specification, designed for describing the behavior of the extended labeling finite state machine, which is convenient for the expression of protocol actions.

2. To verify the SSCOP written in Estelle. Our goal in this part is to develop an environment in which SSCOP can be tested effectively. This environment is designed to test the assured data transfer aspects of the SSCOP. Our testing procedure covers two approaches: "black-box" (user perspective of SSCOP), and "white-box" testing (verification of the protocol itself).

1.3 Chapter Layouts

- In chapter 2, we start by introducing the SSCOP. Some important concepts such as SSCOP functionality, signals, parameters, state diagram and state variables are discussed, so the reader can become familiar with the SSCOP.

- Since we used Estelle to specify and verify the SSCOP, in chapter 3 we concentrate on the concepts of Estelle. The overall concepts of modules, channels, and transitions are covered. As an example, we specify the stop-and-wait protocol in Estelle. Finally, a brief overview of the Estelle Development Toolset (EDT) is given.

- In chapter 4, we present an environment in which the SSCOP is tested. This environment is designed to simulate a scenario similar to the "upper-lower-tester" scheme. The environment is composed of the user modules (as upper layer), the management modules (side layer), and the network modules (lower layer). Our network module is relatively intelligent, and provides us with the basic network features such as packet-loss and transmission delays. In addition, the design of the SSCOP module itself is outlined. The programs for these modules are included.

- In chapter 5, we use the above environment to verify the correctness of the SSCOP from two perspectives: whether the protocol delivers what it promises (black-box testing), and whether it internally works correctly (white-box testing). We use the observers in the Estelle debugger (part of the Estelle Development Toolset) to accomplish this task.

Chapter 2

SSCOP Definition

2.1 SSCOP Overview

The SSCOP is a signaling entity designed to interact with the SSCF, the CP-AAL, and the Management Layer defined in the OSI protocol stack [1][6]. the SSCOP provides a generic reliable data transfer service for different AAL interfaces defined by the SSCF. Two such SSCF interfaces are the Network-to-Network Interface [12] and User-Network Interface[11]. It also provides a peer-to-peer protocol to transfer the information and control messages between any pair of SSCOP entities. The SSCOP uses the services of the CP-AAL to provide the service to SSCF and to the Management layers. The following is a list of functions provided by the SSCOP to its users:

- Transferring of the user data with sequence integrity,
- Error correction by selective retransmission,

- Flow control using window protocol,
- Connection control,
- Error reporting to the manager layer,
- Connection maintenance (keeping connection alive, if necessary),
- Local data retrieval by the user,
- Error detection,
- Status reporting.

The SSCOP can provide both assured and unassured data delivery. Assured data delivery is reserved for the connections requesting for such services (i.e., connection-oriented protocols). However, if the connection is a datagram service, the functionality of the SSCOP virtually map to null, i.e., the SSCOP is bypassed.

2.2 Functions of the SSCOP

The SSCOP performs the following functions.

1. Sequence Integrity. This function preserves the order of SSCOP SDUs that were submitted for transfer by the SSCOP.

2. Error Correction by Selective Retransmission. Through a sequencing mechanism, the receiving SSCOP entity can detect missing SSCOP SDUs. This function corrects sequence errors through selective retransmission.

3. Flow Control. This function allows an SSCOP receiver to control the rate at which the peer SSCOP transmitter entity may send information.

4. Error Reporting to Layer Management. This function indicates to layer management the errors that have occurred.

5. Keep Alive. This function verifies that two peer SSCOP entities participating in a connection are remaining in a link connection established state even in the case of prolonged absence of data transfer.

6. Local Data Retrieval. This function allows the local SSCOP user to retrieve in-sequence SDUs that have not yet been released by the SSCOP entity.

7. Connection Control. This function performs the establishment, release, and resynchronization of an SSCOP connection. It also allows the transmission of variable length user-to-user information without a guarantee of delivery.

8. Transfer of User-Data. This function is used to transfer the user data between users of the SSCOP. The SSCOP supports both assured and unassured data transfer.

9. Protocol Error Detection and Recovery. This function detects and recovers from errors in the operation of the protocol.

10. Status Reporting. This function allows the transmitter and the receiver of the peer entities to exchange information.

2.3 Signals for Layer to Layer Communication

Table 1 represents the signals between the SSCOP and the SSCF (indicated by AA-signals), and the signals between the SSCOP and the Management layer (indicated by MAA-signals). The term "signal" is used instead of "primitive" to reflect the fact that there are no service access points between these layers.

Table 2.1: Signals between SSCOP, SSCF, Management Layers

	Type			
Generic Name	Request	Indication	Response	Confirm
AA-ESTABLISH	SSCOP-UU BR	SSCOP-UU	SSCOP-UU BR	SSCOP-UU
AA-RELEASE	SSCOP-UU	SSCOP-UU Source	Not-Defined	-
AA-DATA	MU	MU SN	Not-Defined	Not-Defined
AA-RESYNC	SSCOP-UU	SSCOP-UU	-	-
AA-RECOVER	Not-Defined	-	-	Not-Defined
AA-UNITDATA	MU	MU	Not-Defined	Not-Defined
AA-RETRIEVE	RN	MU	Not-Defined	Not-Defined
AA-RETRIEVE COMPLETE	Not-Defined	-	Not-Defined	Not-Defined
MAA-ERROR	Not-Defined	Code Count	Not-Defined	Not-Defined
MAA-UNITDATA	MU	MU	Not-Defined	Not-Defined

-The Signal has no parameters

2.3.1 Signal Definition

The following are the definitions of the signals listed in Table 2.1.

1. The AA-ESTABLISH signals are used to establish a point-to-point connection for assured information transfer between peer user entities.

2. The AA-RELEASE signals are used to terminate a point-to-point connection for assured information transfer between peer user entities.

3. The AA-DATA signals are used for assured point-to-point transfer of Service Data Units (SDUs) between peer users.

4. The AA-RESYNC signals are used to resynchronize the SSCOP connection.

5. The AA-RECOVER signals are used during recovery from protocol errors.

6. The AA-UNITDATA signals are used for non-assured, broadcast and point-to-point transfer of SDUs between peer users.

7. The AA-RETRIEVE signals are used to retrieve SDUs submitted by the user for transmission but not yet transmitted by the transmitter.

8. The AA-RETRIEVE-COMPLETE signals indicates that there are no additional SDUs to be retrieved from the transmitter queue.

9. The MAA-ERROR signals are used to report SSCOP protocol errors to the management layer.

10. The MAA-UNITDATA signals are used for the non-assured, broadcast and point-to-point transfer of SDUs between the SSCOP and the peer management layer entities.

2.3.2 Parameter Definition

Table 2.1 lists the parameters associated with each SSCOP signal. The definitions of the parameters are as follows:

1. The Message Unit (MU) parameter is used during information transfer to convey a variable-length message.

2. The SSCOP User-to-User information (SSCOP-UU) parameter is used during connection control to convey a variable-length user-to-user message.

3. The Sequence Number (SN) parameter indicates the value of the transmitter sequence in the received sequenced PDU. It may also used for data retrieval.

4. The Retrieval Number (RN) parameter is used to support data retrieval. This parameter can contain any of the three values: a) "specific value" indicates the retrieval of specific PDU, b) "unknown" indicates the retrieval of the PDUs in the transmission queue, c) "total" indicates the retrieval of all PDUs in the

transmission queue and the transmission buffer.

5. The Buffer Release (BR) parameter indicates whether the transmitter should release the buffers upon the subsequent release of the connection. The value "yes" indicates that the buffers are to be released, and the value "no" means the transmitter will keep the buffers and queues when the connection has been released.

6. The Code parameter indicates the type of the protocol error that occurred (see Appendix A).

7. The Source parameter indicates to the SSCOP user, whether a connection-release is originated from the peer user or it is initiated from the network (in the case of errors). The value "user" indicates that the user initiated the release of the connection, and the value "SSCOP" means that the protocol decided to terminate the connection.

8. The Count parameter indicates the number of sequenced PDUs that so far have been retransmitted. It is used for statistical purposes and later bandwidth assignments.

2.4 SSCOP State Transition Diagram for Signals

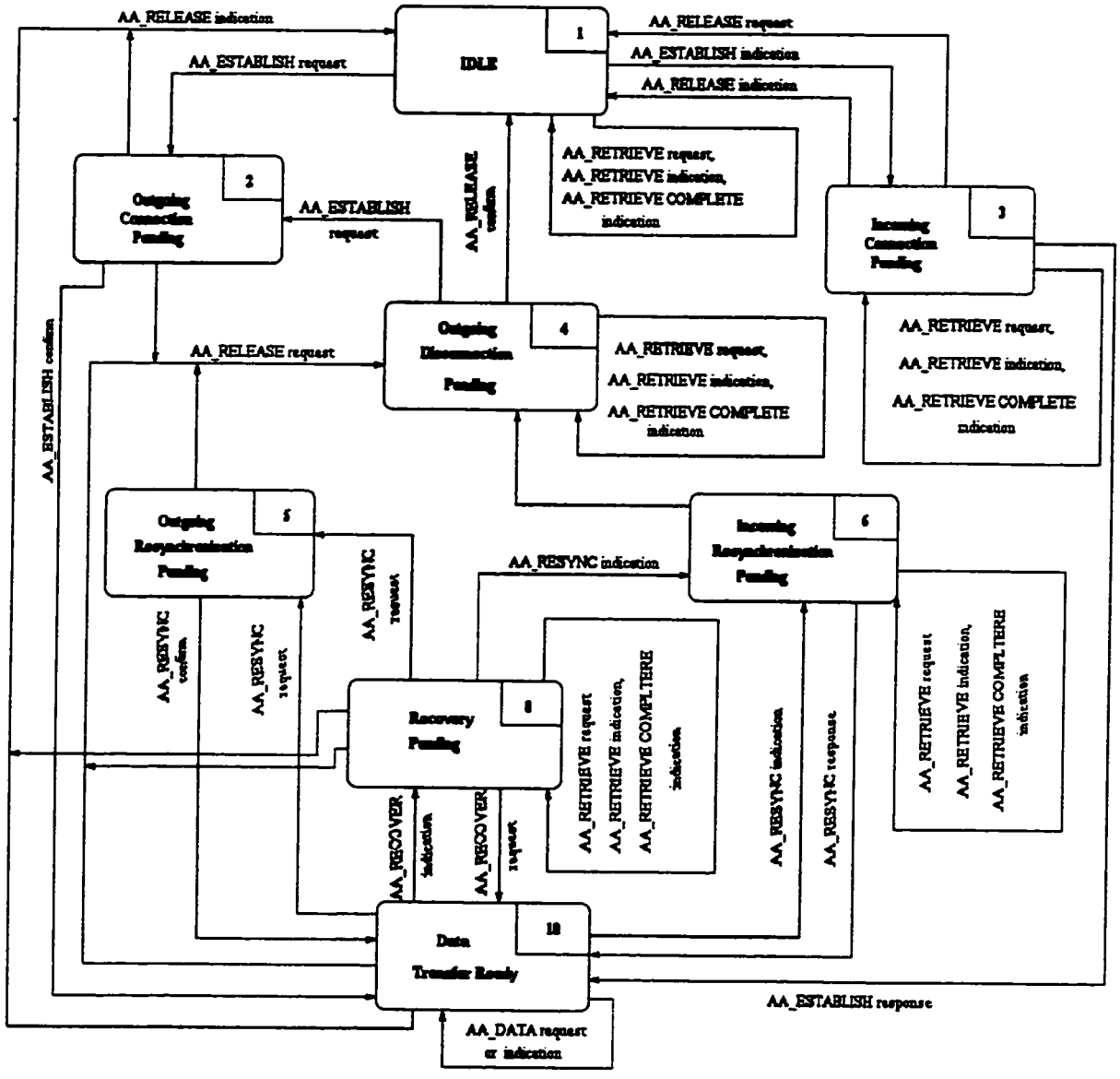
The possible overall sequences of signals at a SSCOP endpoint are defined in the state transition diagram (figure 2.1). The diagram illustrates the behavior of the SSCOP as seen by the SSCF. It is assumed that request or response signals are never issued at same time as the indication and confirm signals. It is also assumed that the signals are serviced immediately. The diagram is adapted from figure 2/Q.2110 [1]. The assumptions are quoted from Q.2110, and are attributes of the System Description Language (SDL), which was used as the primary descriptive tool in Q.2110. These assumptions are appropriate for any single-threaded implementation of the SSCOP, but might not be appropriate for an implementation in a multi-threaded environment.

- The signals AA-UNITDATA.request and AA-UNITDATA.indication are associated with unacknowledged data transfer and thus are permitted in any state;

- Any signal that is not shown in a state, is not permitted in that state;

- It is assumed that no collision of signals between SSCOP and SSCF can occur;

- The idle state reflects the absence of a connection. It is the initial and final of state of any sequence.



Adapted from figure 2/Q.2110[1]

Figure 2.1: SSCOP state transition diagram with respect to SSCF

Note that the states 7 and 9 are not visible to the SSCF.

As the figure indicates, the SSCOP has ten states and their descriptions are the following:

1. Idle state, each SSCOP entity is conceptually initialized in idle state and returns to this state upon the release of a connection.

2. Outgoing connection pending state, an SSCOP entity enters in this state when it requests a connection from its peer. It remains in this state until it receives the acknowledgment from its peer, or its timer indicates that the connection request can not be honored.

3. Incoming connection pending, SSCOP enters in this state when it has received a connection request from its peer and is waiting for its user response.

4. Outgoing disconnection pending, SSCOP enters in this state when it is requesting the release of the connection from its peer.

5. Outgoing resynchronization pending, SSCOP enters in this state when it requests a resynchronization with its peer.

6. Incoming resynchronization pending, SSCOP enters in this state when it has received a resynchronization request from its peer and is waiting for the user response.

7. Outgoing recovery pending, SSCOP enters in this state when it requests recovery with its peer for an existing connection (this state is not visible to the user).

8. Recovery response pending, SSCOP enters in this state when it has completed the recovery, notified the user, and it is waiting for the user response.

9. Incoming recovery pending, SSCOP enters in this state when has received a recovery request from its peer (not visible to the user).

10. Data transfer ready, SSCOP enters in this state when the connection is successfully established, resynchronized or recovered. When both peer entities are in this state, assured data transfer can take place.

2.5 Signals between SSCOP and CP-AAL

The SSCOP makes use of two signals: CPCS-UNITDATA.invoke and CPCS-UNITDATA.signal. SSCOP PDUs submitted for transmission to the peer are placed in the Interface Data (ID) parameter of the CPCS-UNITDATA.invoke. Messages extracted from the ID parameter of the CPCS-UNITDATA.signal are received from the SSCOP peer.

In addition to the above, the CP-AAL also informs the SSCOP about the status of the network. The two signals 'lower layer busy" and 'lower layer very busy" are defined to fulfill this responsibility. The SSCOP must make the needed adjustment to honor these signals.

2.6 SSCOP Protocol Data Units

The Protocol Data Units (PDUs) in SSCOP are defined as follows. The PDU formats are given in Appendix C.

1. Begin PDU (BGN PDU) is used to establish an SSCOP connection between two peer entities.

2. Begin Acknowledge PDU (BGAK PDU) is used to acknowledge the acceptance of a connection request by the user.

3. Begin Reject PDU (BGREJ PDU) is used to reject the connection request of the peer entity.

4. End PDU (END PDU) is used to release an SSCOP connection between two peer entities.

5. End Acknowledge PDU (ENDAK PDU) is used to confirm the release of an SSCOP connection.

6. Resynchronization PDU (RS PDU) is used to resynchronize the buffers and data transfer state variables.

7. Resynchronization Acknowledge PDU (RSAK PDU) is used to acknowledge the acceptance of a resynchronization requested by the peer SSCOP entity.

8. Error Recovery PDU (ER PDU) is used to recover from a protocol error.

9. Error Recovery Acknowledge PDU (ERAK PDU) is used to acknowledge the ER PDU request.

10. Sequenced Data PDU (SD PDU) is a sequentially numbered PDU containing the user information to transfer across an SSCOP connection (assured delivery).

11. Status Request PDU (POLL PDU) is used to request the status information of the peer entity.

12. Solicited Status Response PDU (STAT PDU) is used to respond to the POLL PDU from the peer entity. In SSCOP, each received SD is not acknowledged separately. Instead, a group of SDs are acknowledged via a single STAT PDU.

13. Unsolicited Status response (USTAT PDU) is used by the receiver to notify the sender that some of the PDUs are missing. SSCOP uses this PDU for selective retransmission error recovery.

14. Un-numbered Data PDU (UD PDU) is used for un-assured user information delivery. SSCOP states are not affected by this PDU and there is no sequence number associated with them.

15. Management Data PDU (MD PDU) is used to transfer messages between to peer management entities. Similar to the UD PDU, SSCOP bears no responsibility for the transmission loss.

2.7 SSCOP State Variables

SSCOP defines two sets of variables: a set of variables for the transmitter and a set for the receiver. All variables defined for the transmitter have the acronyms VT(variable-name), the letter "V" stands for variable, and "T" for transmitter. Similarly, all the receiver variables have the acronyms VR(variable-name).

2.7.1 SSCOP Transmitter Variables

1. Send state variable VT(S) is the sequence number of the next SD PDU to be transmitted for the first time (no retransmission). It is initialized to zero and incremented after the transmission of an SD PDU.

2. Poll send state variable VT(PS) is the current value of the poll sequence number. It is initialized to zero and incremented before the transmission of the POLL PDU.

3. Acknowledge state variable VT(A) is sequence number of the next SD PDU to be acknowledged. It is the lower edge of the SD PDU window; it is initialized to zero and incremented upon the acknowledgment of an SD PDU.

4. Poll acknowledge state variable VT(PA) is the poll sequence number of the next STAT PDU expected to be received. It forms the lower edge of the poll window; it is initialized to one and updated by the value of a field carried in the STAT PDU. If an arriving STAT PDU carries a smaller value than the current VT(PA), the error recovery procedure is called.

5. Maximum send state variable VT(MS) is the sequence number of the first SD PDU that is not allowed by the peer (due to the size of the receiver buffer). It represents the upper edge of the SD PDU window. The transmitter shall not transmit any new SD PDU whose sequence is higher than VT(MS). The VT(MS) is updated as the acknowledgments are received.

6. Poll data state variable VT(PD) represents the number of SD PDUs transmitted between two consecutive POLL PDUs, or the number of SD PDUs before sending the first POLL PDU. VT(PD) is reset to zero every time a POLL PDU is sent and incremented when an SD PDU sent.

7. Connection control state variable VT(CC) is the number of unacknowledged BGN, END, ER, and RS PDUs. It is initialized to zero, and incremented when the mentioned PDUs are sent.

8. Connection sequence state variable VT(SQ) is used to allow the receiver to identify retransmission of BGN, ER, and RS PDUs.

2.7.2 SSCOP Receiver Variables

1. Receiver state variable VR(R) is the sequence number of the next in-sequence SD PDU expected to be received. It is the lower edge of the receiver buffer.

2. Highest expected state variable VR(H) is the highest sequence number seen by the receiver plus one. A gap between VR(R) and VR(H) implies that some SD PDUs are missing.

3. Maximum acceptable sequence number VR(MR) is the maximum sequence number that the receiver expects at any given time. It is the upper edge of the receiver buffer. This values is analogous to the VT(MR) at the transmitter. The value of VR(H) can never exceed this value.

4. Receiver connection sequence number VR(SQ) is used to identify retransmission of BGN, ER, and RS PDUs. It is analogous to VT(SQ).

2.8 Queues and Buffers in the SSCOP

Conceptually, two queues and two buffers defined in the SSCOP. The rationale for each of these storage are the followings:

- A circular FIFO transmission queue is defined to store the incoming SDUs arriving from the upper layer(s). As long as its queue has not been filled-up, SSCOP constantly hunts for SDUs submitted for transmission, and stores them in this queue.

- Once a PDU is transmitted, SSCOP keeps a copy of the PDU in a so called transmission buffer for future purposes (i.e., error recovery procedures).

- If a PDU needs to be retransmitted, SSCOP searches the transmission buffer to locate the intended PDU. Having found the PDU, SSCOP inserts it into the retransmission queue for the retransmission. The retransmission queue has a higher priority than the transmission queue.

- When a PDU is received from the network, SSCOP temporarily stores it into a receiver buffer for future delivery to the upper layer(s).

The sizes of these buffers and queues are affected by system resources, network bandwidth, and the speed of the neighboring layers.

Chapter 3

Estelle

3.1 Estelle Description

The Estelle specification language is an ISO standard [14] and is used as a formal specification tool for communication protocols. It is an extension of the Pascal language, designed for the specification of the extended labeling finite state machines [14][17]. Estelle simulates a convenient environment in which one can program and execute the Finite State Machines (FSM). It allows programmers to easily describe both the behavior and the transitions of the FSMs (usually as modules). In addition, Estelle provides channels, which enable different entities (modules) to communicate by means of message passing.

In Estelle, an entity, an independent activity, or an encapsulated task is called a module. A module is described as two parts: the header, and the body. The header of a module defines a set of Interaction Point(s) (IP), which will later be connected to a bi-directional FIFO channel (one channel for each IP). The module uses these IPs to communicate with other modules. The body of the module

defines the needed data structures, functions, procedures, a set of states, and a program (called transitions), which explains the behavior of the module for the given states.

When a module receives a message from its channel, depending on the state of the module and the message type, it activates the corresponding transition(s). The activated transition(s) will later be scheduled for execution. Based on the module type, the process of scheduling may be deterministic or non-deterministic (explained below).

In this chapter, we concentrate on the concepts such as channel, module, transition, and scheduling in Estelle, so we become more familiar with this environment. As an example, we also included a complete specification of the "stop-and-wait" protocol in Estelle in Appendix D.

3.2 Channels In Estelle

A channel is the communication link between two entities. Modules send messages to each other through the channel between them. In Estelle, channels are FIFO, and bi-directional, with an infinite capacity. Each channel carries only a set of pre-defined messages. Figure 3.1 shows a channel definition in Estelle, which simply specifies the role of the channel for each IP. As the figure indicates, there are certain types of messages that flow into the channel. Furthermore, the definition of a channel splits into two major roles. Since a channel connects two IPs, each IP (one at each end) plays a different role in the channel.

```
Channel channel-name (role1, role2);
    By role1:
        message1: message_type_1;
            ...
        messagek: message_type_k;
    By role2:
        messagem: message_type_m;
            ...
        messagen: message_type_n;
```

Figure 3.1: An Estelle Channel Definition

For example, assume that module A in figure 3.2 has a declared IP "s", which assumes role1. Similarly, module B has an IP "r", which assumes role2.

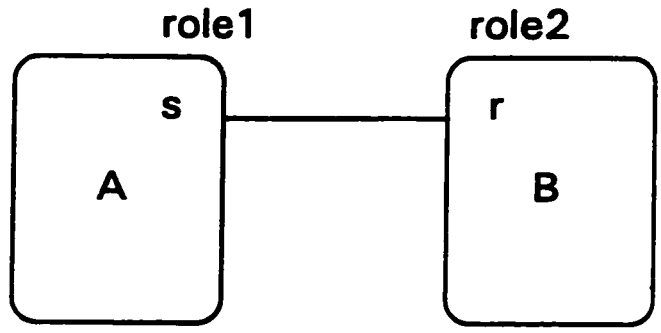


Figure 3.2: A sample connection in Estelle

Since A.s plays role1, it can only send those messages whose types are defined in the role1 part of the channel definition (messages 1 through k, figure 3.1). B.r plays role2, so it can send those that are defined in the role2 part (messages m through n). B.r receives the messages sent by A.s, and A.s receives the messages sent by B.r.

When a message is placed on the channel, it is stored in a queue associated with the destination IP, and is immediately visible to that IP, i.e., no delay is imposed for message transmission. As mentioned earlier, the capacity of a channel is infinite. This assumption is made so that the applications need not concern themselves about the number of messages stored in the channel.

3.3 Modules in Estelle

Figure 3.3 shows an outline of the Estelle description of a module. A module is a separate entity, an instance, an independent activity, or a well-defined task. As figure 3.3 indicates, the definition of a module contains two major parts: the header, and the body. The header of a module defines some general information about the module. It is responsible for declaring the module's type (described below), the Interaction Point(s) (IP) and the role of the module for each IP (i.e., sender, receiver,...). After the keyword "IP", each interaction point is defined as follows

ip-id : channel-id (role-id);

where ip-id is the interaction point variable, channel-id is the channel identification type (described in section 3.2) and role-id is the role of the module for that interaction point (ex., sender or receiver).

The header of a module may also define the import and export variables, if needed. The import variables are defined as variable names followed by their types. The export variables are declared using the keyword "export".

```
MODULE A Module-Type (import var.);
        IP X: channel-name (sender);
           Y: channel-name (receiver);
        Export variable(s);
END;
BODY AA For A;
      { const / var / func. / proc. }
      { other nested modules }
   STATE   s1,s2,...;
   INITIALIZE TO initial-state;
   Begin
       { extra initilization variable }
   End;
   Trans
       { set of transitions }
END; { body }
```

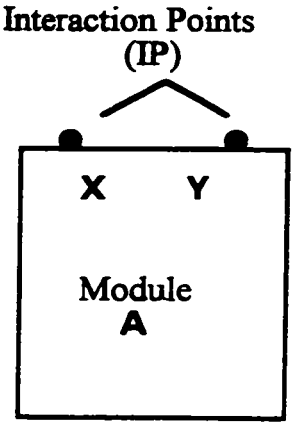


Figure 3.3: Estelle Module Definition

The body of a module defines the specifics. It defines the data structures, functions and procedures needed for the module's operation. Declaration and definitions of data structures, functions and procedures are similar to Pascal except for the use of dynamic allocations. That is, the use of pointers is restricted in Estelle; pointers can not be used to access objects beyond the scope of the module. The body of a module may also contain the definition of another module(s)

(entirely). In this case the nested module is considered to be a child module. Child modules may share the parent IP to communicate with the rest of the world.

The body of a module also defines a set of states (optional) as the following:

STATE state-identifiers;

where "state" is a keyword and the state identifiers are any PASCAL identifiers. The keyword "initialize" is used to initialize the module to its initial state.

Finally the body of a module defines a set of transitions (described in section 3.4). The idea is to break down the program into some independent actions. The collection of these actions describes the behavior of the module.

Before terminating this section we need to discuss about the module types. In Estelle modules are given some predefined types (figure 3.3). The type of the module identifies a class of the modules with some specific attributes. These attributes: systemactivity, systemprocess, process and activity, are designed for the scheduling purposes in Estelle's environment. Estelle uses this information to schedule the enabled transitions of the module. For example, when a module defined as systemactivity, Estelle non-deterministically selects only one enabled transition of the module whereas, for a systemprocess module, enabled transitions are scheduled in a synchronized fashion.

3.4 Transitions In Estelle

Transitions are the actual break down of the FSM program; they are the internal module behavior. Each transition is defined as two parts: clause-group, and transition-block (figure 3.4).

```
FROM state-identifier TO state-identifier
        When  IP-identifier(message-name)
        Provided  (boolean expression)
        Priority  (expression)
        Delay  (expression)
        Name  transition-name
        Begin
            {body of the transition}
        End;
```

Figure 3.4 : Transition Definitions in Estelle

Collectively, a clause-group defines the firing condition(s). A clause-group may consist of one or more of the clauses shown in figure 3.4. It should be noted that the delay clause and the when clause cannot be used together in a transition header.

- **From** clause, indicating the state in which the transition may fire.

- **To** clause, is used when a transition starts from one state and ends in another state.

- **When** clause, indicates that when a message is received from an interaction point, the transition may fire.

- **Provided** clause, represents a Boolean expression.

- **Priority** clause, represents the firing priority for the transition.

- **Name** clause, identifies the transition name.

- **Delay** clause, indicates the amount of time that the activated transition must delay before firing (ex., used as timer)

A transition is said to be activated (or enabled) when its clause-group evaluates to true predicate(s), except for the "TO" clause. Activated transitions are then scheduled for the execution on the behalf of the module.

The transition-block is the body of the transition, and its statements are similar to Pascal. In addition to the Pascal statements, Estelle also allows an "OUTPUT" statement, so the modules are able to send out messages through their Interaction Points.

For example, considering figures 3.1, 3.2 and 3.4 the transition

```
From establish
When  A.s (message2)
Provided Not (full (queue))
Name store-in-queue:
Begin
        enqueue (message2);
        output A.r (messagem);
End;
```

represents, when the module A is in establish state, and the message1 is received from the interaction point "s", provided that queue is not full, the transition inserts the message in the queue and outputs the message-m (ex. acknowledgment message).

3.5 Other Estelle concepts

In this section we describe some other concepts such as building timers, scheduling and execution of transitions in Estelle. At the end of this section, we also provide a complete example to show the process of protocol specification using the Estelle language.

Transition Scheduling in Estelle

The system executes in a succession of computation steps. Each computation step of the system begins by non-deterministic selection of one (in case of a systemactivity module) or several (in case of a systemprocess module) transitions among those transitions offered by each module. That is, in each step, the active modules are consulted to name their enabled transitions. The Estelle scheduler selects one or several enabled transition (depending on the module type) from each module. The selected transitions are then executed in parallel. A computation step ends when all of them have been completed. The system is said to be deadlocked if in a computation step no module offers any enabled transition to the scheduler.

Transition Execution in Estelle

In Estelle, the execution of a transition is said to be atomic and takes no time. During the execution of a transition, the system is not interruptable and thus no logical clock time is consumed. This decision prevents any sudden modification to the group-clause of the currently executing transition. As the result, the integrity of the executing transition cannot be compromised.

Timer Transitions in Estelle

In the specification of a protocol we often need to use timers. To fulfill that need, Estelle simulates an internal logical clock, which ticks every some predefined

amount of time. Programmers use this feature to define timers for their specification, for example:

```
From establish
Delay (10)
Provided no_acknowledgement_received
Name timer-expiry
Begin
      "resend the previous packet"
End;
```

implements a timer (at state establish) to retransmit the previously sent packet after delaying 10 clock ticks provided that no acknowledgment is received in the mean time. During this delay another transition may fire (if it receives acknowledgment) and set the 'no-acknowledgment-received" flag to false, causing the clause-group of the timer transition to be false and thus be deleted from the scheduler at the next computation step.

The value for delay is application dependent and is usually the estimated round-trip time. It should be set such that there is enough time for receiver to forward its response to the transmitter.

Primitives in Estelle

Primitives are functions and procedures defined outside of the specification module. Primitives are analogous with the external functions in C or Units in Pascal. In Estelle, Primitives are designed to give the user a flexibility to define

subroutines specifically for a particular interest, regardless of the restrictions in Estelle. For example, similar to other specification languages, Estelle is not designed to interact with the user. Thus, it contains poor library functions to handle the user interaction. Consequently, one can write improved input and output routines (perhaps in C) and notify Estelle to link them.

In order to notify Estelle about the existence of the external routines, they must be defined as primitives in the Estelle specification. Once the primitives are defined, Estelle allows its modules to call these routines as a regular function call. The format for defining the primitives is the following.

> **Function** identifier (variables and their types): return-type; **Primitive**;
> **Procedure** identifier (variables and their types); **Primitive**;

Primitives are visible to all modules and thus they may not be defined inside a module. We have used the primitives in the SSCOP specification for the reading and writing from/to the files, and for pointer manipulation using the C language. The need to use this feature is shown in the example provided in Appendix D.

3.6 Estelle Development Toolset

The Estelle Development Toolset (EDT)[15] (see figure 3.5) includes the Estelle Compiler (Ec), the Estelle Simulator/Debugger (Edb), and the Universal Test Driver Generator (UTDG). Ec translates the Estelle specification into C language source code, after a complete static analysis (lexical, syntactical and semantic). Edb builds a meta implementation of the specification and allows interactive and extensive simulation either randomly or with predefined scenarios[16]. The user may control, observe, and trace the execution of the specification by the means of the simulation. The UTDG builds automatically a simple simulation environment (not used in our work).

Our verification process for the SSCOP uses the Edb to set the observers that can report any error in the protocol simulation. We discuss the observers later in chapter 5.

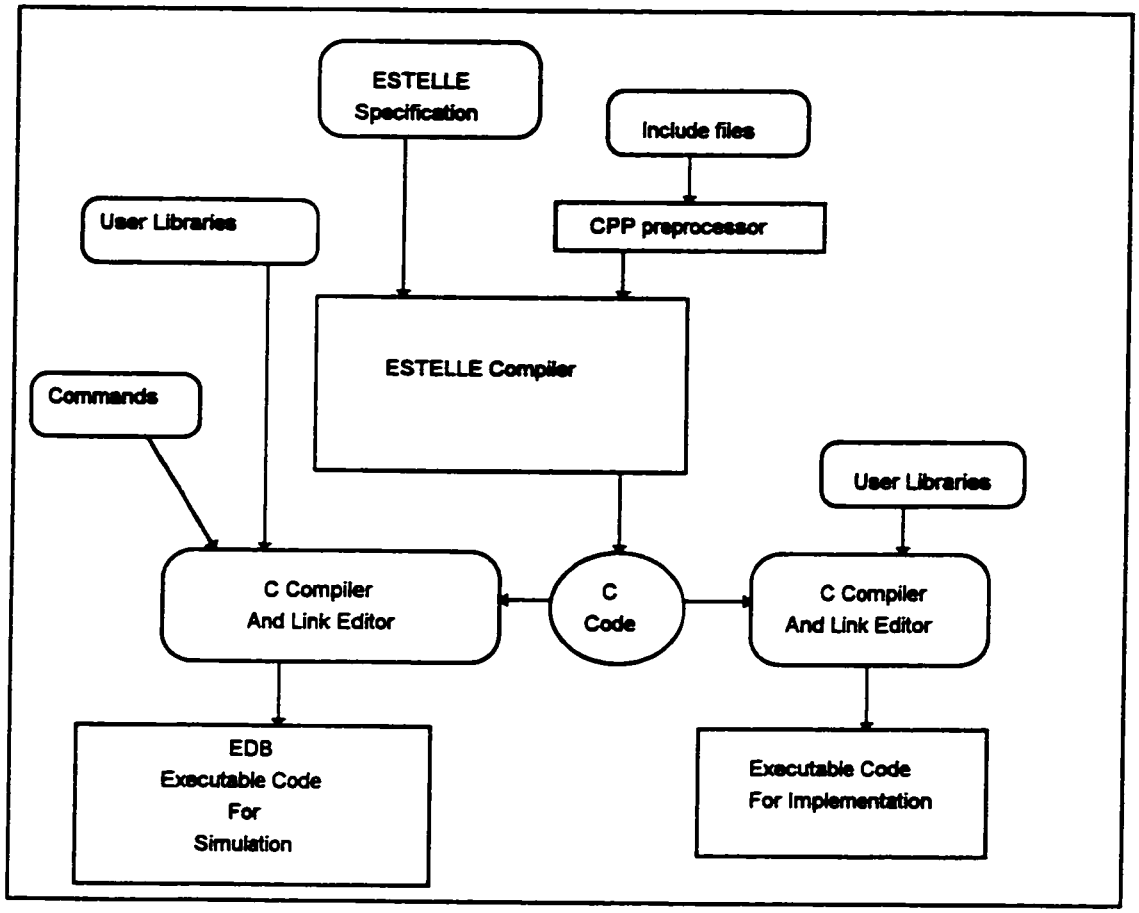


Figure 3.5: Estelle Development Toolset

Chapter 4

The Simulation Model

4.1 Model Description

As indicated in figure 1.1 (chapter 1), SSCOP is surrounded by two other layers: SSCF and CP-AAL. Since every layer in the OSI stack for the ATM communicates with its Management Layer [18], thus we can infer that SSCOP is also surrounded by the Management layers. In fact, in section 2.2 (table 2.1) we observed that SSCOP defines two signals for the management layer. Consequently, one should consider the effect of these layers in modeling an environment for the SSCOP.

Figure 4.1 represents a scenario in which the SSCOP should be tested. In this module each of three neighbors of the SSCOP is an independent module--the user module represents the SSCF, and the network module represents the CP-AAL layer of the AAL. Our management module is simply collects the received signals.

In the next four sections, we discuss the behavior of the above modules and

explain their functionality. These modules are designed so that the whole Estelle specification simulates a well-suited environment that one can use to test the SSCOP.

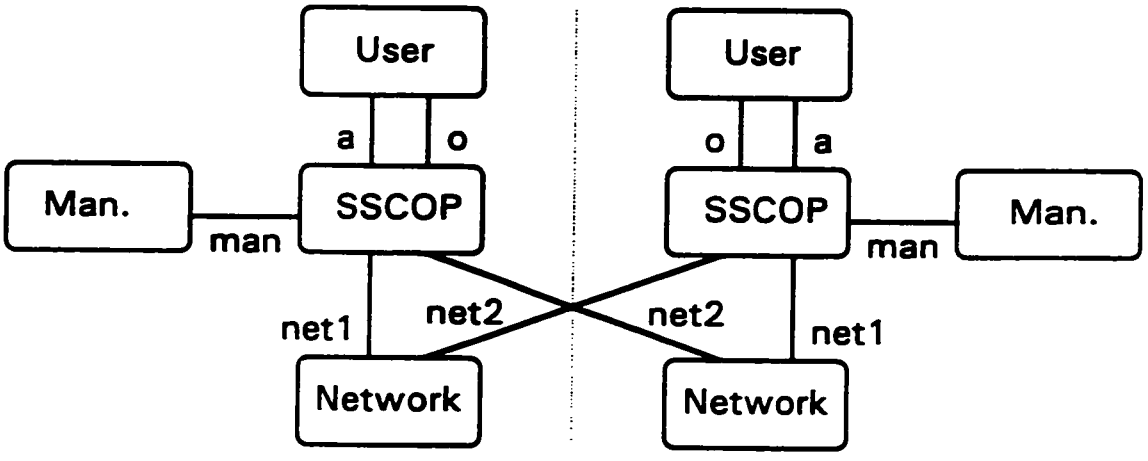


Figure 4.1: The Simulation Model

4.2 User Module

The user module is a generic module designed to invoke the SSCOP data transfer routines. It is responsible for simulating the connection request, the production of data-packets, and finally the disconnection request (see figure 4.2). To produce data, the user module generates two types of messages: assured and un-assured messages. Assured messages are produced in establish state. They must be delivered (by SSCOP) in-sequence and their integrity must be preserved whereas un-assured data packets have no such restrictions. As far as the user

module is concerned, it must send each type of message to its proper channel. Assured messages are directed to the "a" (stands for assured) channel, whereas other messages are sent to the "b" channel. The user module is also responsible for receiving these messages, and in this case, simply displays them. The process of sending the messages is the following:

1. Starting from the "idle" state (the initial state), user module requests a connection from the SSCOP,

2. If the request returns successful, go to step 3, otherwise, after a fixed delay go to the step 1 (fixed delay is used to allow the request be processed by other layers),

3. Once the "connection established" indication is received, move to state "establish" and generate an establish response signal,

4. Generate a new message (possibly read from a file) and do the following:

   - if the message is to be assured, send it through the assured channel, otherwise send it to the other channel.

   - if there are no more data to be sent go to step 5, otherwise go to 4.

5. Send a signal indicating the end of session, go to state "discontinue" and wait for a graceful termination.

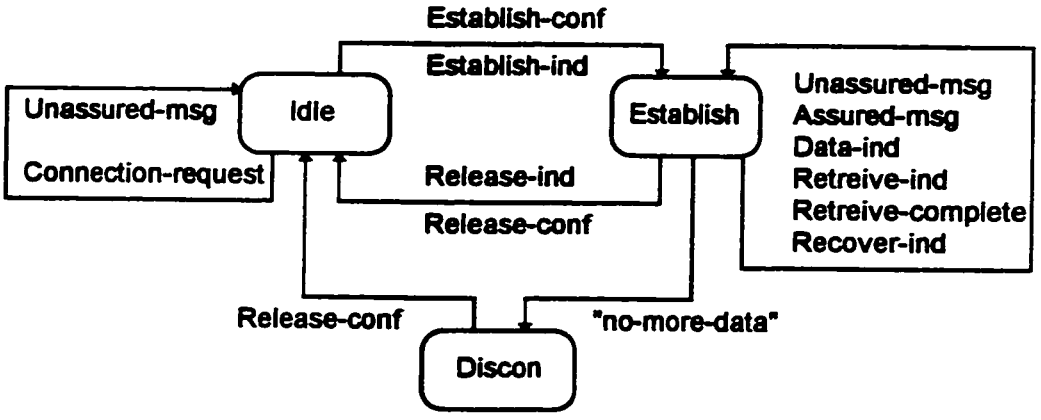


Figure 4.2: User Module State Diagram

In parallel with the above activities, the user module must watch for incoming signals (either from its peer or from the SSCOP) to take an appropriate action[2]. For example, during the data transfer session user module may detect the arrival of a signal indicating that the connection is no longer alive; it must end its activity and take a proper action. Figure 4.3 depicts the list of signals handled by the user module. This corresponds to table 2.1 of this thesis.

It should be noted that there are two types of signals in the above figure. The first type is the events triggered locally (ex., establish-conf), and the second type is the events triggered remotely (ex., establish-ind).

---

[2] The user module ignores signals arriving at states where they are not defined.

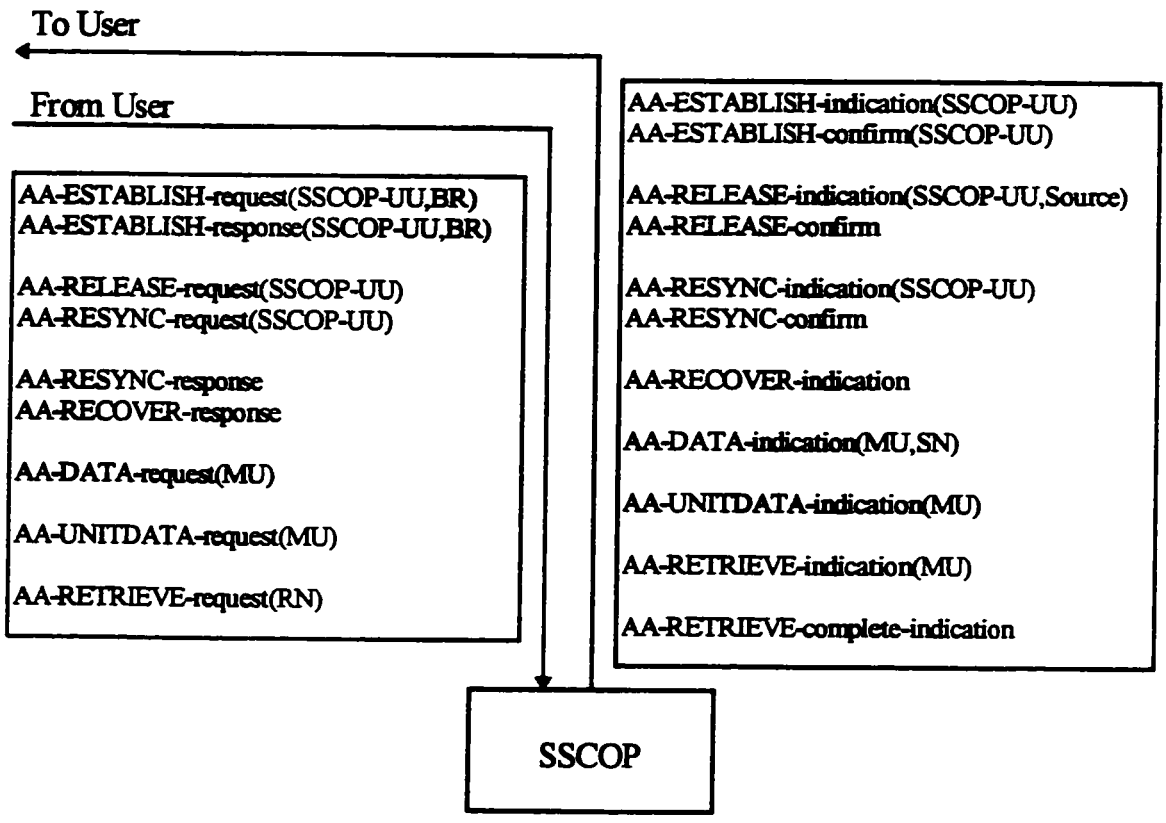


Figure 4.3: Signals between SSCOP and User Module

The complete Estelle specification of the user module is listed in Appendix E.

4.3 Management Module

The management module is responsible for monitoring the data transfer procedure for each of the SSCOP connections. It is designed to receive the signal

(MAA-ERROR) from the SSCOP whenever some unexpected event occurs. The collected information is stored in a data-base and used to help the manager module to make a correct decision for any situation. For example, when the SSCOP detects that a packet is missing it requests the retransmission of the packet from its peer, and also it informs its local management layer. If the manager module realizes that the rate of packet loss is unreasonably high, it may decide to abort the connection in one case, or decide to increase the bandwidth in another case. These decisions are based on the type of service (QoS), the state of the network, and the past history of the connection. For our implementation, the manager module makes no attempt to abort or increase bandwidth of SSCOP connections.

In addition to the above, management layer may wish to negotiate with its peer entity. Thus the management layer is designed to invoke the SSCOP for data delivery (see table 2.1). The SSCOP treats the management-data as unassured data messages. Figure 4.4 depicts the signals handled by the management module.

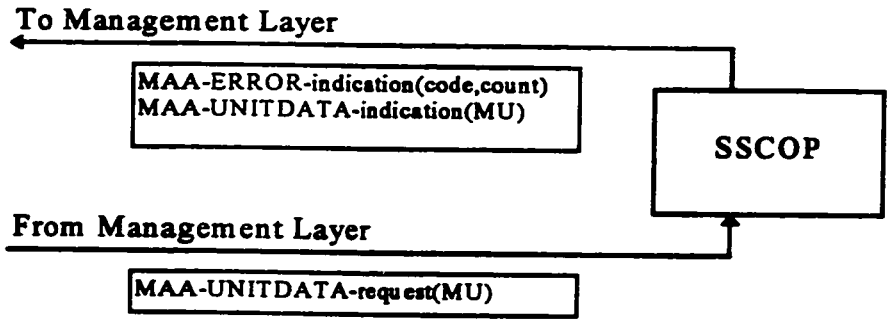


Figure 4.4: Signals between SSCOP and Manager

In this simulation, the management module simply displays the signals received from the SSCOP. It also periodically sends some information to its peer to simulate the act of negotiation. The Estelle program for the management module is listed in Appendix F.

4.4 Network Module

An ATM network is designed in such a way that it can drop cells, or insert cells, but will never re-order cells. In addition the network will introduce some delay.

The network module in our simulation is emulating the services provided by the CP-AAL: delivery of (re-assembled) SDUs. Because of the properties of the underlying ATM network, the network module must exhibit two properties: SDU loss and SDU delay.

The network module (in figure 4.1) is responsible for receiving SDUs from SSCOP and forwarding them to its peer. The way in which this process can take place is critical to our work of verification of the SSCOP. The network module is designed to occasionally fail forwarding some packets (packet-loss simulation). That is, the network module randomly select some packets and fails to deliver them to the other SSCOP entity. We expect the error detection and error recovery procedures of the SSCOP to be triggered by this action. In addition, the network module is also designed to delay forwarding SDUs to trigger the SSCOP's timers and dynamically sets the processing rate of the SSCOP. We now examine these behaviors in more detail.

4.4.1 Structure of the Network Module

As mentioned above and shown in the figure 4.1, the network module acts as a bridge to deliver SDUs between two SSCOP entities. It has two IPs namely: net1 and net2. It also has an internal queue for outgoing messages (see figure 4.5). Messages arriving at net1 are put into the outgoing queue, to be delivered some time later to net2.

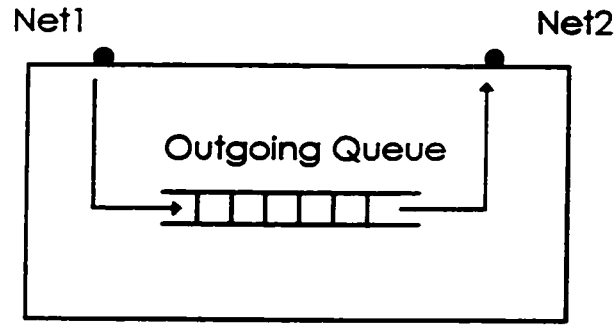


Figure 4.5: Structure of Network Module

It is important to mention the significance of the outgoing queue. Although the existence of this queue imposes an extra indirection (net1 → queue → net2), it gives us a flexibility to control the rate of the message processing and message loss at the network (explained below). Our network module is designed to simulate two major behaviors, namely: packet-loss and packet-delay.

4.4.2 Packet Loss Simulation

Arrival of a message on the Interaction Point net1 activates a transition that stores it into the outgoing queue. The transitions form a set, with individual transitions each handling a different packet type. Every transition that receives a message from net1 calls a random generator function, which returns a value from zero to ten. If the outcome of the random number is below a pre-defined value (called the loss-rate), the transition ignores the message (without forwarding); otherwise it stores the message into the outgoing queue for further delivery. For

example, let us assume that the loss-rate value is set to 2, and the outcome random number has the value 1. In this case the transition terminates immediately; had the value been greater than or equal to 2, the transition would have put the message into the outgoing queue. Notice that in this example, the rate of the packet-loss is about 20%. Also note that any change to the pre-defined value of the loss-rate results in a different rate of packet loss.

In our module we define a set of loss rate variables (data loss-rate, control loss-rate) to independently control the message dropping of the network for different type of packets. For example, the loss rate of the data-packets (SD PDU) is different from the rate of the control-packets (RS PDU, ER PDU). Having lost the outgoing packets as mentioned above, helps us to observe two important properties of the SSCOP: 1) if a data type packet is dropped at the network, we expect to see the SSCOP initiate a USTAT PDU (defined in section 2.6) requesting that the lost packet be retransmitted[3]; 2) if a control packet is lost, we anticipate that the SSCOP will patiently wait for the next control packet. The waiting time is bounded by the SSCOP timers.

---

[3] Note that the SSCOP is a protocol with the selective retransmission scheme

4.4.3 Network Delay Simulation

As noted in the previous section, a set of transitions are in charge of receiving messages arriving at net1 and storing them into the outgoing queue (figure 4.5). Another set of transitions are responsible for draining the outgoing queue and forwarding the messages to net2, as follows. When a message appears in the queue, the transitions delay[4] for some pre-defined value (called base net-delay) before forwarding the message to net2. That is, the network module does not immediately process the message upon its arrival (similar to real networks). The introduced delays cause the outgoing queue to grow. When the length of the queue is sufficiently large (above a certain threshold), the network module sends a signal (called lower-layer-busy) to its local SSCOP indicating that the network is overloaded, and the SSCOP lowers its speed. The network module informs the SSCOP to increase its speed when the length of the queue falls below the threshold.

The above mechanism helps us to study the SSCOP behavior in the presence of a heavily loaded network. It also let us verify the SSCOP timers with respect to the round-trip delays. SSCOP should honor the 'lower-layer-busy" signal and attempt to reduce its processing rate. Failing to do so, SSCOP experiences a high

---

[4] Timer transitions are discussed in section 3.5

rate of packet loss and packet delays, which leads to aborting of the connection by the management layer.

In our simulation, the values for "net-delay" and "loss-rates" are pre-defined, but in reality these values are established as part of the negotiation for a Quality of Service [10]. The values used for loss-rates are much higher than the values that would be experienced in a real network. This is intended to provide a very hostile environment, and to reduced the processing time for the simulations.

4.5 The SSCOP Module

The ITU-T final draft recommendation Q.2110 [1], which describes the SSCOP, consists of a few pages of English text, and 50 pages of SDL diagrams. The SDL diagrams have primarily been used as a framework in the Estelle specification of the SSCOP. The process of translating the SDL diagrams into Estelle is rather lengthy and challenging, because every detail must be considered. Graphical languages such as SDL tend to concentrate more on the data flow, to give the user the needed tool for visualizing the protocol. On the other hand, Estelle relies heavily on the detailed definitions, and it operates on the data structures and algorithms. As the result, one must follow a consistent scheme for the translation process.

The translation process started by examining the SDL diagrams and after a careful examination of the diagrams, the following concepts had to be addressed, in detail.

1. The SSCOP channels need to be completely designed.

2. Major data structures such as buffers and queues are constantly referred to, but never defined. Thus we asked the following questions: What should these structures be like? What are the necessary fields?

3. The SDL diagrams use five different timers for the SSCOP. How should these timers be defined? How they should be incremented? Are transitions with delay clauses sufficient for the describing the timers?

4. How one can handle the multiple branching shown in the diagrams?

While the first two concepts are related to the data structure issues, the other two are algorithmic and thus the design must be influenced by the properties of Estelle. We addressed the above concerns with the help of the original documentation of the draft recommendation (English text).

4.5.1 Channels of the SSCOP

Figure 4.1 suggests that the SSCOP requires 5 different channels, each of which carries multiple signals. The definitions of these signals determine the way in which these channels must be defined (described in section 3.4). For example,

considering the figure 4.3 and the table 2.1, the "b" channel between the SSCOP and the user is defined as follows:

```
{ channel for other SSCF to SSCOP signals
(separate channel facilitates priority treatment) }

CHANNEL sscfXsscop_other (role_sscf, role_sscop);

  BY role_sscf :
     { unassured data request }
     aa_unitdata_req (mu : data_t);

     { data retrieval request }
     aa_retrieve_req(rn : retrieve_sdnum_t);

     { connection establishment request }
     aa_establish_req (sscop_uu : info_t; br :
               boolean);

     { connection establishment repsonse }
     aa_establish_res (sscop_uu : info_t; br :
               boolean);

     { connection release request }
     aa_release_req (sscop_uu : info_t);

     { connection resynchronisation request }
     aa_resync_req (sscop_uu : info_t);

     { connection resynchronisation response }
     aa_resync_res;

     {recover response}
     aa_recover_res;


  BY role_sscop :
     { unassured data indication }
     aa_unitdata_ind (mu : data_t);

     { retrieval indication }
     aa_retrieve_ind (mu : data_t);

     { retrieval complete indication  }
     aa_retrieve_complete_ind;

     { connection establishment indication }
```

```
aa_establish_ind (sscop_uu : info_t);

{ connection establishment confirmation }
aa_establish_con (sscop_uu : info_t);

{ connection release indication }
aa_release_ind (sscop_uu : info_t; source :
          source_t);

{ connection release confirmation }
aa_release_con;

{ connection resynchronisation indication }
aa_resync_ind (sscop_uu : info_t);

{ connection resynchronisation confirmation }
aa_resync_con;

{ recovery indication }
aa_recover_ind;
```

Using the above approach, we define all needed channels and their data types.

4.5.2 Data Structures of the SSCOP

As mentioned in the section 2.8, the SSCOP requires two queues and two buffers. The data structures for them are designed based on the complete examination of the SDLs and the English text of Q.2110.

- The transmission and retransmission queues need to store the user data and the sequence numbers (MU, SN). Therefore we define these queues as arrays of a structure containing MUs and SNs, with the procedures to manipulate them (ex., enqueue, dequeue,...).

- The transmission buffer needs to store the transmitted data with their respective time stamps, VT(PS), for error recovery purposes. Given that reading an item from the transmission buffers always implies moving it to the retransmission queue, we define this new array with the following properties:

    1. reading from this array is destructive (implicit delete).

    2. The insert and search algorithm are standard.

- The receiver buffer is similar to the transmission queue except for the implementation functions. Thus, we add the necessary adjustments.

Additional data structures and variables can be similarly discovered from the SDLs and the English text of Q.2110.

4.5.3 Timers for the SSCOP

Once a timer starts, it runs in parallel with other activities in the module. Modules need three basic operations from the timers. They need to start, restart (reset), and stop the timers dynamically. Implementing a timer by using a transition with a delay clause as described in section 3.5, ensures the ability to start and stop the timer when it is needed. That is, the timer starts running when its provided clause become true, and it stops when the provided clause becomes false. Thus, we only need to explore how to reset a timer. Consider the following

example: we would like to reset the timer 'no-response" [5] whenever a STAT PDU is received (no-response timer is described in chapter 5). Defining this behavior by the following transition:

```
From DataTransfer
When "a STAT PDU arrives"
Name restart-timer:
Begin
     "set the provided clause flag of the
          timer to false"
     ...
     "set the flag to true"
end;
```

will fail to reset the timer, because the execution of a transition is atomic. This transition assumes that by setting the provided clause of the timer to false and then to true, the timer will restart. However, Estelle does not interrupt the execution to evaluate the current status of the transitions (for which the timer transition is a part). The next evaluation for the enabled transitions starts after the current transition(s) are fully executed.

To solve the above problem, we need to define two transitions to implement such a timer. Consider the following:

---

[5] A timer that is always active in the data transfer state.

```
Provided ((TickCount ≥ MaxDelay) and flag)
Name timer expiry:
     Begin
        "Actions to be taken"
     end;


Provided ((TickCount < MaxDelay) and flag)
Delay (SomeConstant)
Name clock-is-ticking:
     Begin
        TickCount := TickCount + SomeConstant;
     end;
```

where the flag is the original condition for which the timer was designed, MaxDelay is the maximum amount of time that should be delayed, and the TickCount is a variable.

To start a timer, the value of the flag is set to true and the TickCount to zero. Setting the flag to false stops the timer. However, the resetting of a timer can now be easily implemented as:

```
From DataTransfer
When "a STAT PDU arrives"
Name restart-timer:
Begin
   TickCount := 0;
end;
```

Since the value of the flag is never changed, the timer is still running, but it starts counting again from zero. The timer expires when the summation of the TickCounts becomes greater than MaxDelay.

4.5.4 Branching issue for the SSCOP

Consider the following SDL diagrams:

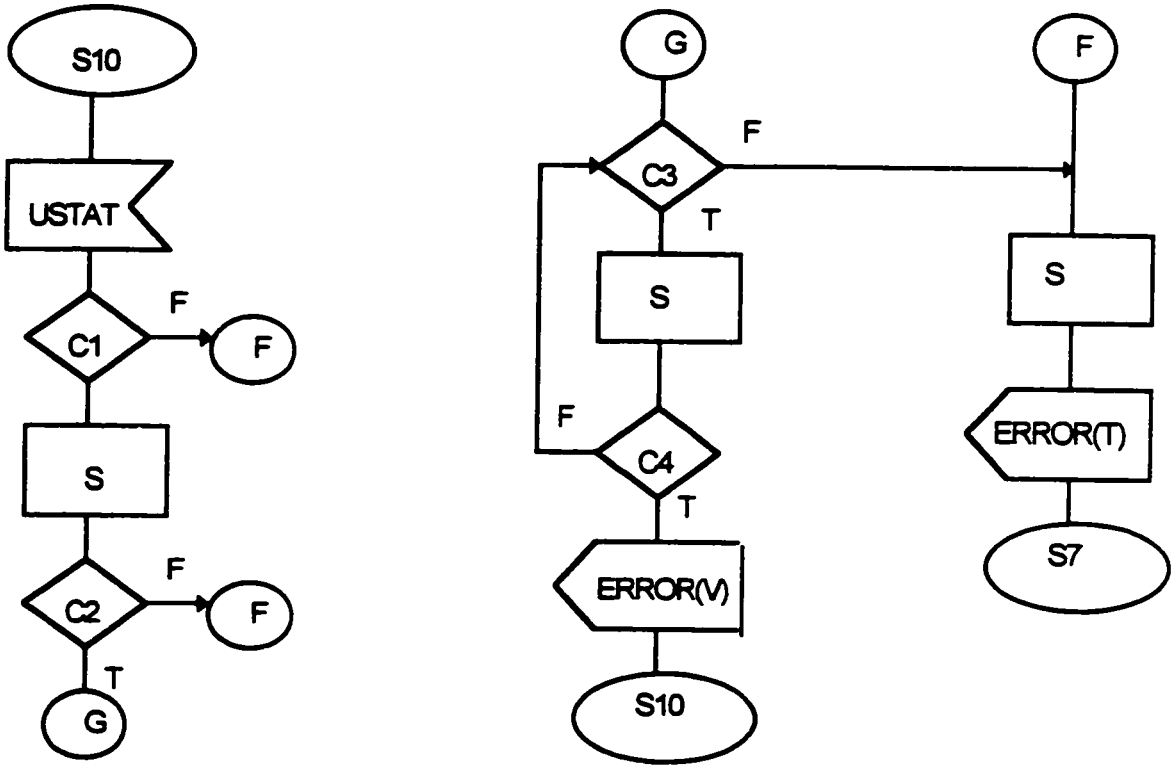


Figure 4.6: An Example of SDL diagrams

The above SDL diagram states that when the USTAT PDU is received, a sequence of tests is necessary to determine whether the current state should be changed. If any of the tests C1, C2, or C3 fails, the SSCOP enters state 7. Otherwise the SSCOP remains in the same state (S10) and continues its operations. The real issue is that there are multiple possible branches in the diagram where the tests may fail.

In Estelle, the decision for state change must be stated prior to the transition body (that is, in the transition header). Therefore, one cannot define a transition such that the body of the transition states that when a test fails, move to a different state. Clearly, we need two transitions. One transition remains in the same state to cover the G path. The second transition however, changes the state of the module (upon its execution) to cover the F path. As the result we have two transitions with the same when clause but different provided clauses as shown below:

```
From S10
When "a USTAT PDU arrives"
Provided (successful_Tests())
Name Ustat-is-received-without-error:
Begin
    "Path G"
end;
```

```
From S10 To S7
When "a USTAT PDU arrives"
Provided (NOT successful_Tests())
Name Ustat-is-received-with-error:
Begin
    "Path F"
end;
```

When the USTAT arrives, if the successfullTests function returns true, the first transition becomes active. Otherwise the second transition becomes active.

The above function must perform all the tests shown in the SDL diagram before it returns any result. Since one of the conditions (C3) is in the loop, the function must predict the loop outcome up to its last iterations. Therefore, a clear understanding of the loop functionality in the SDL diagrams becomes absolutely essential. We may be able to simulate the loop (as the above case), or we may want to determine the loop invariant to predict its outputs. The later strategy is particularly helpful when the test condition gets modified in the loop.

Similar to the above strategy, we identified all the needed paths and branches and defined the needed transitions, data structures and Boolean functions.

Finally, it should be mentioned that in the SDL diagrams, the ignored signals at each state are not shown. These signals must be defined in Estelle. They are defined by describing a transition whose header has the needed conditions to catch the signal, but whose body is empty.

Chapter 5

Verification of SSCOP

5.1 Basic Concepts

The process of verification is to examine whether a product is correct and delivers what it promises. The way in which the verification procedure should take place is an unsettled debate and depends on economic factors and the complexity of the product. In the literature, there are different approaches that one can use to address the issue of verification [21]. At one end of the spectrum is the exhaustive test technique. This technique is based on constructing test routines that explore all possible scenarios (or states), to study the behavior of the product. At the other end of the spectrum is the traditional approach of the functionality test (testing from the user point of view), which builds specific scenarios to examine the product for the intended requirement(s). Other verification techniques such as heuristic based testing and decomposition of the functionality testing reside between the above two approaches.

In the context of the protocol verification (SSCOP in particular), the exhaustive verification techniques, though they are complete, are often not practiced due to economic reasons and the "state explosion". The SSCOP is inherently complex and large (contains lots of state). Building test cases to cover all the possibilities and scenarios is a costly and time consuming approach. Although there are some tools to automatically generate the test cases (for the given specification), their effectiveness is limited. In addition, exhaustive verification methods often fail due to "state explosion", when applied to a complete specification of a real life protocol [20].

The traditional approach of the protocol verification, which based solely on the user service interface, is also not adequate since it can only detect failures in the achievement of the intended services, without proving the correctness.

5.2 The Proposed Test Cases for SSCOP

Our verification method for SSCOP focuses successively on individual functions to reduce the complexity of verified properties. To verify the correctness of the SSCOP functionality (mentioned in section 2.2), we simulate test cases to observe the behavior of SSCOP in performing its task. These test cases are designed to look for anomalies in each of the SSCOP properties. The intention is to show that SSCOP delivers its responsibilities in both the "clean" and the "hostile" environments.

The so called "clean" environment refers to an ideal situation where no undesirable events such as packet-loss can occur in the underlying network. The network module as described in section 4.4 is capable of constructing such an environment because the rate of packet loss is under the user control. Simulation for this scenario helps us in verifying the SSCOP user service interface (black-box testing). As far as the user is concerned, SSCOP must deliver (in sequence) the submitted data packets.

A "hostile" environment, on the other hand, is a real world situation where networks can arbitrary delay or even fail to forward the packets. With the help of the observers (explained in the next section), we can create an even more hostile environment by tampering with the SSCOP variables. These simulations are also necessary to give us more insight into the SSCOP behavior in the presence of

unexpected events (white-box testing). For example, the selective retransmission mechanism of the SSCOP is tested every time a data packet is lost.

In section 2.2, we listed a series of functionality that the SSCOP must honor. Some of these functions are concerned with the user point of view (ex. functions 1,4,6,8), while others are related to the negotiations between two SSCOP entities. The first set of functions must be tested with respect to the user. That is, for this set of functions, the SSCOP should be viewed as a 'black-box" (section 5.4). The remaining functions however, concern the SSCOP behavior and thus should be tested as such, that is, as a "white-box" (section 5.5).

It is important to note that the above method is not designed to test for the performance of the SSCOP. The lack of the ATM hardware support prevents us from analyzing the efficiency of the SSCOP functionalities. Consequently we focus on the correctness of the SSCOP.

5.3 Using Observers for Verification

In Chapter 3 we introduced the Estelle Development Toolset [16][19] and mentioned that Edb allows us to observe, control and trace the execution of the specification by the means of the simulation commands. The 'observer" commands in Edb make it possible to describe certain situations that the user

wants to observe or detect and, at the same time, they allow defining associated action(s) to be taken. For example, when an anomaly is detected the simulation can be stopped.

A group of observers can be chained together to concurrently observe different scenarios in the execution of the specification. Observers can be enabled, disabled, and even deleted dynamically. After each computation step (defined in section 3.5), Edb executes the sequence of the enabled observers. The use of these observers is the main tool for the implementation and verification of simulated scenarios.

The verified function is modeled as a black box with an interface where its activity can be observed. The interface of the system is the set of observed objects. They typically include a subset of state variables and the interaction points of the module involved. Similarly, the internal behavior of the protocol can be verified using a different set of observers.

Observed simulation for repeated runs on a set of *relevant* scenarios, with random variation of some situation parameters, can thus be a framework for verification of the specification.

In our verification process we use the observers in four different ways, namely: reporters, controllers, managers, and modifiers. Some observers simply watch for particular events to occur and report them. Some observer are set to break the simulation in the case of a fatal errors (controlling the execution). Manager observers are responsible for dynamically disabling and enabling the other observers. Finally, we use also the observers to modify the data structures while the execution is in progress (explained below).

5.4 Black-Box Testing

The two user and management modules view their local SSCOP as a 'black-box". As figure 4.1 indicates, there exist a user module and a management module at each side of a connection in which they operate symmetrically.

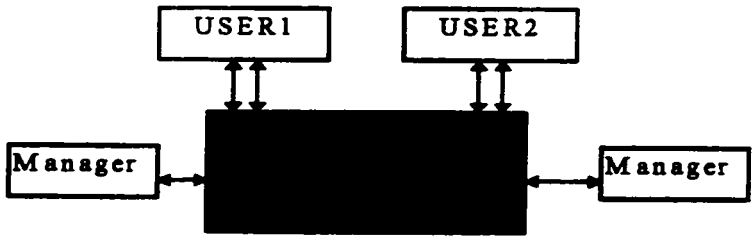


Figure 5.1: User view of SSCOP

Figure 5.1 abstracts the view of the user and management modules from their underlying layer(s). They regard the underlying layer as a 'black-box" and expect a certain functionality from it. Users want the SSCOP to respond to the connection requests, to deliver data (either assured or unassured) and to notify them of resynchronizations. As a result, the following set of actions are programmed into the user module, to help us in verifying the expected services:

- Request for a connection, when there is none,
- Respond to or refuse a connection request, when there is one pending,
- Produce data (assured or unassured) and invoke the SSCOP to deliver them,
- Upon the reception of data:

- if data are unassured, display them,

- otherwise, check that the sequence numbers of the received data are in order,

• Request for release when there are no more data to send,

• Respond immediately to the error-recovery and resynchronization signals.

Similar to the users, the management modules view the SSCOP as a 'black-box". They anticipate that the SSCOP will report any unexpected behavior in the data flow. They also invoke the SSCOP data transfer procedures to deliver unassured data messages. Since delivery of the management data is unassured, we do not consider it here.

We verified the above functionalities as follows. For the different scenarios, at the time of the execution, we used the commands of Edb to create some observers for verification of the intended services with respect to the user and manager modules.

To verify the sequenced data delivery, we set an observer to watch for the sequence of the incoming data packets at the receiver module (ex., user2 is the receiver module). The observer is responsible for comparing the sequence number of the previously received packet with the current sequence number. If an out-of-sequence condition is observed, the observer breaks the simulation and issues the appropriate message.

A second observer is set to watch for the arrival of the AA-RESYNC-indication and AA-RECOVER-indication at the sender module. When these signals arrive, the observer displays a message indicating that the SSCOP error recovery procedure has been invoked.

Finally the third observer is set to observe the events at the management modules and reports the received signals. As mentioned earlier, Appendix A depicts the error coded signals for the management module. For example, error code "V" represents the reception of the out of sequence packets. Using those, we can identify the type of error that has been reported to the management module.

5.5 White Box Testing

Similar to the previous section, the network module in conjunction with the Edb observers is the basis for verification of the SSCOP behavior. Together, they cooperate to simulate scenarios that the SSCOP must handle. These scenarios are

drawn from the SSCOP functionalities and include the following:

- Error correction by selective retransmission,
- Flow control,
- Keeping alive the connection
- Error recovery and status reporting

We will show that our approach can verify these functionalities. Let us first explain the needed information for the following figures.

The figures contain four columns. The action and delivered columns are self explanatory. The "Tx" column represents the transmitter side and the "Rx" columns represents the receiver side. At the transmitter side, the data packets are shown as two values: the VT(S) and the VT(PS).

The control packets such as POLL PDU is sent with fields: VT(S), and VT(PS). The STAT PDU is sent with four fields: VR(R), the received VT(PS), the buffer size, and the list of the missing PDUs. The list of the missing PDUs is formatted as " {2,3}" indicating that the 2 is missing and the 3 is present.

5.5.1 Error Correction by Selective Retransmission

When a sequenced data packet is lost, SSCOP must detect the missing packet and perhaps issue a USTAT PDU to the transmitter requesting the missing PDU. In SSCOP, the transmitter periodically requests the receiver status using the POLL PDU (figure 5.2). The receiver responds to the status request and issues a STAT PDU carrying its current state variables and a list of the missing PDUs. By using the USTAT PDU the receiver is also given the responsibility to immediately inform the transmitter about the missing PDU(s), without waiting for a POLL PDU (figure 5.3).

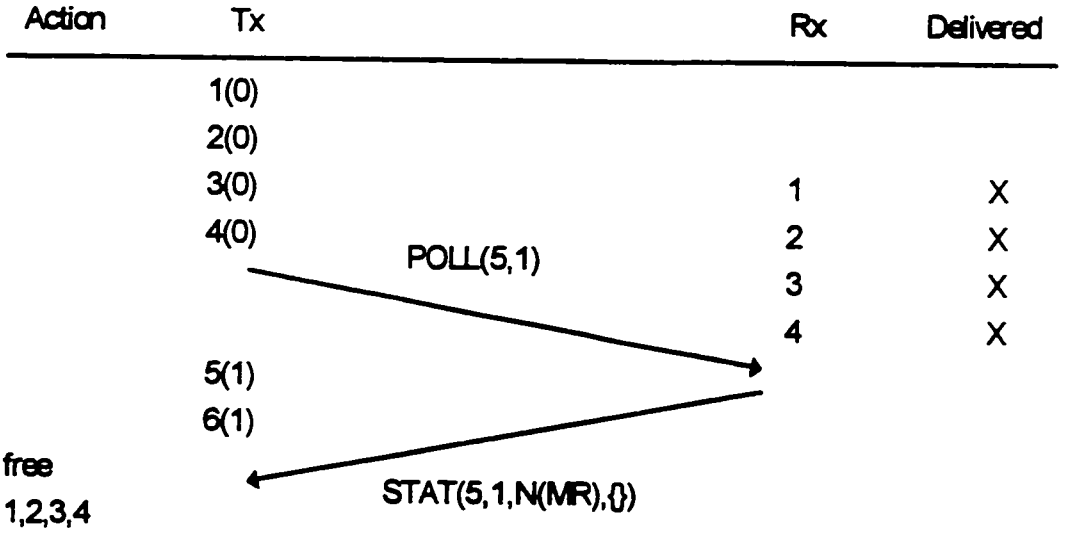


Figure 5.2: Error Free Operation

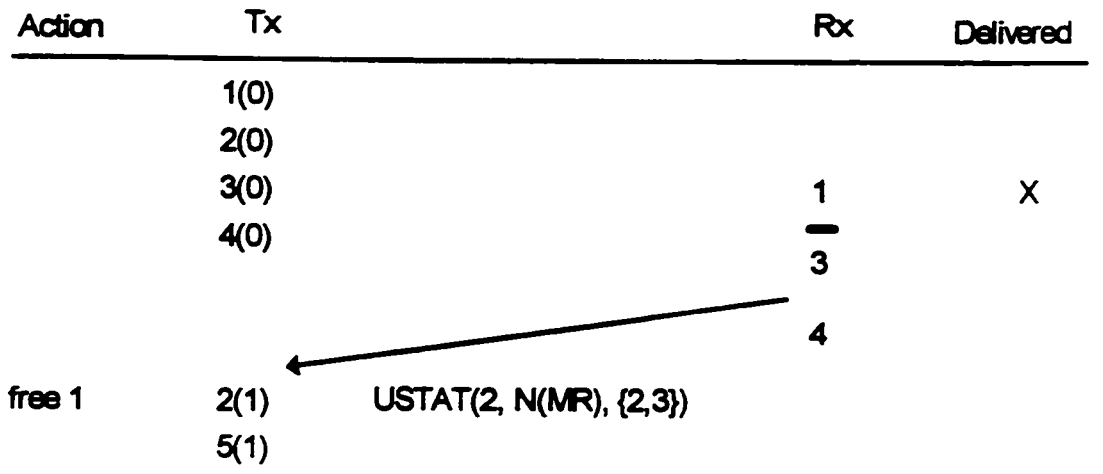


Figure 5.3: Error Correction Via USTAT PDU

To verify the SSCOP retransmission mechanism, we need to ask the following: A) Does the receiver SSCOP correctly detect the missing PDUs? and B) For the scenario depicted in figure 5.4 where there is an overlap for the reported missing PDU, how many times does the SSCOP attempt to retransmit the PDU?

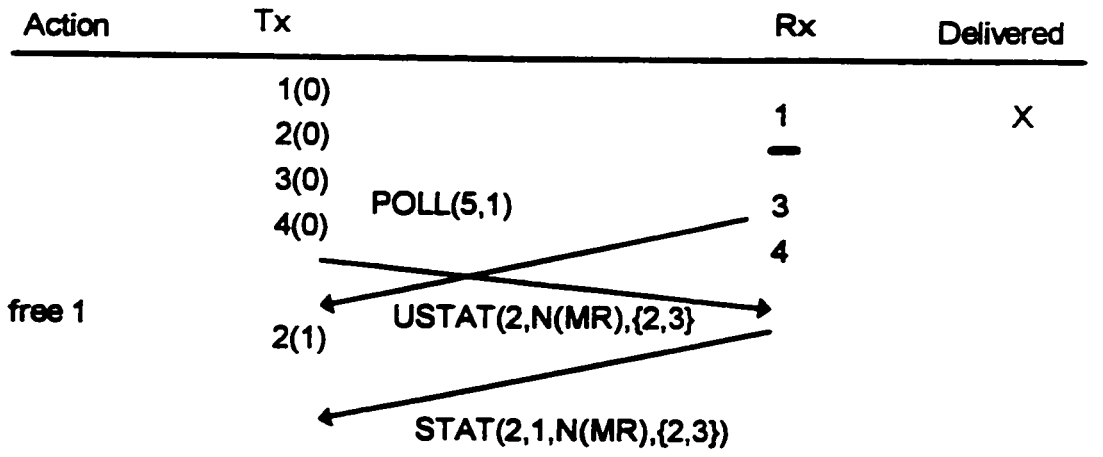


Figure 5.4: Error Detection of the SSCOP

The first question was partially answered in the previous section where we tested the SSCOP as a black box. However, in this section we try to determine if the SSCOP indeed issues the USTAT PDU whenever the next expected PDU is missing. It is important to note that if the SSCOP had failed to issue the USTAT PDU, our black box test would have still been successful since the SSCOP recovers itself through the POLL and STAT PDUs. Therefore, we need to test (specifically) for the USTAT PDU. We accomplish this by setting an observer in Edb to watch for the following. When a received SD has a sequence number greater than the VR(H), the VR(H) is less than VR(MR) and SSCOP does not issue a USTAT PDU, the observer breaks the simulation and displays the appropriate message.

To answer the above second question (part B), we consider two possibilities. Let us assume that a USTAT reported a missing packet and the SSCOP has inserted it into the retransmission queue. When a STAT PDU carrying the same information arrives, the first possibility is that the missing SD is still in the queue, and the second is that the missing SD has already been retransmitted.

For the first case, we set an observer to search the retransmission queue for the repeated sequence number. Since every SD that is about to be retransmitted is inserted into the retransmission queue, the above observer is effective to report any double insertion anomaly.

As for the second case, the SSCOP simply ignores the STAT report because, the reported missing SD has now a VT(PS) greater than or equal to the N(PS) carried by the STAT. Every time an SD is transmitted or retransmitted, the SSCOP updates the current poll sequence number of the SD in the transmission buffer. Therefore, in this case, since the SD is already retransmitted (its VT(PS) is updated), SSCOP safely ignores this retransmission request. No observer is necessary for this action.

5.5.2 Flow Control

The notion of flow control in the context of the SSCOP is divided into local and peer-to-peer flow control. Local flow control is concerned with long queuing delays in the lower layer(s). As mentioned earlier, our network module generates

the signal 'lower-layer-busy', which carries a Boolean expression set to true when the length of its queue is sufficiently large. The SSCOP must temporally suspend its data transfer (MD, UD, SD PDUs) to give priority to the control messages. To verify this, we enable an observer upon the reception of the busy signal, and disable it when this signal indicates that the network is no longer busy. During its life time, the observer monitors the SSCOP and it expects no transmission of data packets.

The peer-to-peer flow control is responsible for defining an operating window for the SSCOP. This window can shrink and grow dynamically[6]. The transmitter sets the lower edge of the window by VT(A). The receiver supplies the upper edge value using the N(MR) field in the BGN, BGAK, RS, RSAK, ER, STAT and USTAT PDUs. Sudden changes to this value by the receiver, will perhaps result in loss of SDs in transit; no verification is necessary.

5.5.3 Keeping the Connection Alive

In the case of the prolonged absence of data transfer, the SSCOP is responsible for keeping the connection alive, if the users wish to do so. As long as the users do not issue the "AA-RELEASE-request" signal, the SSCOP keeps the

[6] The size of the window can never shrink below the VR(H) - VT(A).

connection alive by periodically sending and receiving control packets (POLL and STAT PDUs).

To keep the connection alive, the SSCOP defines two separate timers: 'no-response" and 'keep-alive". The 'keep-alive" timer is armed only when there is no SD to be transmitted (or retransmitted). The timer 'no-response" however, is always armed in the data transfer state. The timer 'no-response" is longer than the "keep-alive".

When the timer 'keep-alive" expires, the SSCOP sends a POLL PDU to signal the peer that the connection is to remain alive. The peer responds through the STAT PDU. Upon the reception of the STAT PDU, both timers are restarted and so the process continues. Failing to receive the STAT PDU, allows the 'no-response" timer to expire and forces the SSCOP to abort the connection.

The functionality of the 'keep-alive" timer is to periodically remind the SSCOP to generate the POLL PDU. The function of the 'no-response" timer however, is to define the absolute interval that the SSCOP can tolerate without any notification from its peer. In practice, SSCOP should tolerate a few 'keep-alive" timer expiries without the response from the peer.

We can verify the behavior of the "no-response" timer by two strategies. The first is to set an observer to watch for the status of the "keep-alive" timer. As soon as the timer is armed, the observer disarms it by modifying the Boolean expression of the PROVIDED-clause of the respected transition (see the example of the timer transitions in section 3.5). As the result, the SSCOP sends no POLL PDU (since there is no one to remind it). It is clear that the expiry of the "no-response" timer is imminent.

The second strategy also sets an observer to watch for the "keep-alive" timer to be armed, but, it modifies the control "loss-rate" values (in particular poll-state-loss-rate). Modifying that value to ten will result in 100% packet loss. Therefore, no STAT PDU can be transmitted, causing the "no-response" timer to expire.

It is worthwhile to mention that the above observers are watching for the "keep-alive" timer to become active. The verification for the correctness of this timer is then implicit. That is, if the timer never activates, the observer fails to proceed.

5.5.4 Error Recovery and Resynchronization

Any sequence number problem that confuses the SSCOP will result in error recovery activation. For the data transfer, the SSCOP has the intelligence to discover the gaps and reports them as they appear (using the USTAT PDU). This is the first attempt of the SSCOP toward error recovery. As was seen in the

previous section, the SSCOP is also capable of recovering itself when a USTAT PDU is lost (through the POLL and STAT PDU). These two steps accomplish a powerful and efficient way to inform the transmitter of the packet loss. However, a complication may arise. For example, if a SD is received with the sequence number smaller than VR(R) (below the sequence number that the receiver expects), the SSCOP invokes the error recovery procedures and it moves from the state of the data transfer to the error recovery state.

Error recovery routines assist the SSCOP in recovering from the protocol errors relating to the assured data transfer. The error recovery is completed when the users are notified, the transmission buffer is cleaned (if necessary) and finally all the transmitter and the receiver state variables are reinitialized. We have imposed scenarios that force the SSCOP to activate the error recovery procedures, enabling us to study the error recovery processes.

It is important to mention that without external help, we cannot trigger the error recovery states of the SSCOP. Our model is capable of arbitrarily dropping packets. However, this is not a sufficient condition for the SSCOP to see any major problem such as protocol errors, for the following reasons.

- When a SD is lost, SSCOP behaves as was previously explained (figure 5.3).

- When a control packet is lost, SSCOP patiently waits for the next control packet. As long as the SSCOP timers allow, the SSCOP tolerates the loss of the control packets[7].

A more hostile environment is needed to invoke the SSCOP error recovery procedures. Therefore, we use the observer intervention to corrupt the SSCOP's state variables, thereby forcing the SSCOP to enter the error recovery states.

According to the SSCOP specification, there are mainly two ways that the receiver SSCOP activates the error recovery procedures. The first situation is when there is any conflict in the operating window (SD window). As was noted earlier, a SD carrying a sequence number lower than the VR(R) implies that the sender and the receiver are out of sequence and they must be recovered. The second reason for the error recovery is when there is an inconsistency in the poll window. That is, a STAT PDU carries a poll sequence that is smaller than the VT(A).

With the help of the observer we can occasionally tamper with the SSCOP state variables. An observer can be set and left disabled for a period of time (ex., 100 firing steps) until it is enabled by a different observer. The enabled observer can now modify the state variables and later be disabled by another observer.

---

[7] Loss of a few consecetive STAT PDUs will expire the "no-response" timer, forcing the SSCOP to cease the connection.

Reasonable candidates for state variable changes are: VR(H) to be reduced, VR(R) to be increased, VT(PA) to be increased, VT(S) to be reduced and finally VT(A) to be increased. Modification to any of these variables eventually leads to the corruption of the SSCOP windows (SD, or POLL). The confused SSCOP is then forced to activate the error recovery procedures.

Once the state error recovery is entered, one can use the observers to monitor the behavior that follows. The specification of the SSCOP indicates that the SSCOP must still accept the "AA-DATA-request" signal at this state (state 7). The SSCOP connection timer must be armed. Users must be notified. Finally, upon the successful error recovery the transmitter buffer is flushed, if necessary, and the SSCOP state variables are reinitialized. Methods for verification of the above functionalities are similar to the methods that we have been using in the previous sections. For example, to verify the SSCOP connection timer we use the same strategy as the 'keep-alive" timer, by not allowing any connection packets to transmit.

The SSCOP specification regards the issue of the resynchronization to be invoked by a user driven signal, "AA-RESYNC-request". Users may need to be resynchronized with one another. Receiving this signal, the SSCOP suspends its data transfer and enters in the resynchronization state. To verify this issue, we need to modify our user module to simulate this scenario by adding a few transitions. A timer transition (defined in section 3.5) is set when the user module

enters the establish state. When the timer expires, the transition requests resynchronization and causes the SSCOP to enter into the associated state. The needed verification can proceed from this state.

5.6 Results of the Simulations

It was said at the beginning of the chapter that our intention was to show that the SSCOP delivers its functionalities in both "clean" and "hostile" environments. We have had different simulation runs in both of these environments, with and without observers. The results are outlined below. A breif summary of the results is shown in tabular form in Appendix G.

5.6.1 Black Box Simulation Results

Our first scenario started by setting all the loss-rate values (defined in section 4.4.2) to zero, simulating a "clean" environment for the SSCOP. In this simulation the SSCOP correctly delivered the data packets and no error was reported by the observers. This verifies the correctness of the SSCOP in the above environment.

In the second attempt, the data loss-rate value is modified to set the rate of the data packet loss to about 20%. No error is reported by the observers at the user modules. The observer for the management module however, reported a series of

signals with the error coded "V", indicating that the SSCOP was experiencing some packet loss and that it had requested the retransmission of those packets from its peer. We explain more about this behavior in the next section.

Our last scenario included the simulation of a more "hostile" environment by allowing all different loss-rate values to be above zero percent. As the result, all different packets (data or control packets) were randomly dropped at the network with some probability. This simulation was also successful in the sense that no out-of-sequence packet was observed. There was no reason for SSCOP to invoke error recovery (that is, SSCOP did not change its state to error recovery state). However, as in the previous simulation, the management observers were busy reporting the received error codes.

The lesson drawn from these scenarios is that the SSCOP was successful in delivering the assured data packets with respect to its users, and notifies them for any error recovery. Connection requests were honored, and data messages were sent as expected. The SSCOP was also able to communicate with its management layers and report the unexpected events. This concludes our view of SSCOP as a "black box".

5.6.2 White Box Simulation Results

Reporting to the management layer started when we

switched to the less friendly environment. As long as the SSCOP experienced the packet loss (control, or data), it vigorously recovered itself through the retransmission or POLL-STAT PDUs. The periodic 'timer-poll" expiry ensures that the receiver SSCOP is often polled to give the transmitter the information needed for advancing the operating windows (SD and POLL). Actions taken by the SSCOP were accounted for and they verified successfully.

We visited and verified the functionality of the major states including the error recovery and resynchronization. We had to corrupt the SSCOP data structures in order to step through its error recovery mechanism. The SSCOP timers were also tested and their functionality verified.

Looking at the number of reported error messages and the firing steps in Edb suggest that the fewer control packets result in a smoother and faster data flow. Although the control packets are vital to the SSCOP self recovery, their excessive use in fact tend to generate more delays and processing time. By increasing the poll and 'ho-response"timer values, we were able to operate more smoothly at the expense of the larger queues.

Chapter 6

Conclusion

The final ITU-T draft recommendation, Q.2110, offers the Service Specific Connection Oriented Protocol (SSCOP), which is used to support the signaling entity above the ATM-Adaptation Layer. This protocol is responsible for the assured data delivery service and provides an environment that the upper layer applications need for their signaling mechanism. We fully specified the SSCOP in Estelle, and used the specification to verify the functionality of the SSCOP.

The Estelle Development Toolset provides us with a flexible environment for the verification of the protocol. As the result, we developed a model (an environment) described in Estelle to verify the SSCOP. The model consists of two User modules acting as the upper layers, two Management modules operating as the side layers and finally two Network modules acting as the layers below the SSCOP. The network modules were designed to arbitrary delay and randomly drop the packets. With the help of this module and the observers defined in Edb (Estelle debugging tool), we simulated different scenarios and situations that the SSCOP should be tested in.

In the verification process we verified the functionalities of the SSCOP as both a 'black-box" and a "white-box". We used the 'black-box" testing to verify the SSCOP service interface (assured data delivery). The "white-box" is used to study and verify the behavior of the internal operation of the SSCOP. We verified that the SSCOP indeed behaves as it is specified.

Finally, we presented the results of the simulations and claimed that the SSCOP correctly delivered the assured data messages. Our rigorous test scenarios verified the service interface and the internal behavior of the SSCOP in both friendly and hostile environments. Major states were visited and the results were satisfactory.

References

[1] ITU-TD PL/11/-20C, Final Draft Recommendation Q.2110, 'B-ISDN - ATM Adaptation Layer--Service Specific Connection Oriented Protocol (SSCOP)".

[2] TA-NWT-001113, "Asynchronous Transfer Mode (ATM) and ATM-Adaptation Layer Protocol Generic Requirements", Bellcore, issue 1, August 1992.

[3] CCITT Recommendation I.350, 'General Aspects of Quality of Service and Network Performance in Digital Networks, Including ISDN", COM XVIII-R 114-E.

[4] T1E1.2/92-020, 'Broadband ISDN Customer Installation Interface, Physical Media Dependent Specification", ANSI Draft Standard, Feb. 1992.

[5] CCITT Recommendation I.371, 'Traffic Control and Congestion Control in B-ISDN", Geneva 1992.

[6] CCITT Recommendation X.210 "OSI layer service conventions".

[7] Stassinopoulos, G. "ATM Adaptation Layer Protocols and IEEE LAN Interconnection". Proceedings, 15th Conference on Local Computer Networks, October 1990.

[8] CCITT Document TD-XVIII/10 "AAL Type 5, Draft Recommendation text for section 6 of I.363", 29 January 1993, Geneva.

[9] ITU-TS draft Recommendation Q.93B 'B-ISDN User-Network Interface Layer 3 Specification for Basic Call/bearer Control". May 1993.

[10] Glenn Estes, et al. "ATM User-Network Interface Specification", The ATM Forum, September, 1993.

[11] ITU-T Recommendation Q.2130 'B-ISDN Signaling ATM Adaptation Layer--Service Specific Coordination Function for Support of Signaling at the User-to-Network Interface (SSCF at UNI)".

[12] ITU-T Recommendation Q.2140 'B-ISDN Signaling ATM Adaptation Layer--Service Specific Coordination Function for Signaling at the Network Node Interface (SSCF at NNI)".

[13] CCITT Recommendation I.363 "B-ISDN ATM Adaptation Layer Specification".

[14] Estelle - "A Formal Description Technique based on Extended State Transition Model", ISO International Standard 9074, 1989.

[15] S. Budkowski, Estelle Development Toolset. 'Computer Networks and ISDN

Systems Journal", Special issue on FDT Concepts and Tools, vol.25, no.1, 1992.

[16] S. Budkowski, P. Dembinski, L.Lumbroso, 'User Guide for the BULL Estelle Simulator/Debugger (Edb)", DRCG/ARS Internal Report, October 1988.

[17] HASEGAWA et al, "Automatic ADA Program Generation from Protocol Specification Based on Estelle and ASN.1", Proceedings of the 9th International Conference on Computer Communication, Tel Aviv, October 1988.

[18] William Stallings, "ISDN and Broadband ISDN", 1992.

[19] S. Budkowski, et al. 'Estelle, Simulator/Debugger (EDB), User Reference Manual", Institute National des Télécommunications, Départment systèmes et Réseaux 9, rue charles Fourier, 91011 Évry, France.

[20] G. Holtzman, "Design and Validation of Computer Protocols", Prentice Hall, 1991.

[21] Carlo Chezzi, Mehdi Jazayeri, Dino Mandriolo, 'Fundamentals of Software Engineering", Prentice Hall, 1991.

Appendix A

A number of events will cause errors to be indicated to the layer management entity. The following is the list of these error codes and the states affected by them. This list is taken from the Table A.1/Q.2110 in [1].

Table A.1: Error Codes and Related Signals

Error Type	Error Code	Error Condition	Affected States
Receipt of Unsolicited or Inappropiate PDU	A	SD PDU	1,3,6,9
	B	BGN PDU	6,7,8,9
	C	BGAK PDU	1,3,6,7,8,9
	D	BGREJ PDU	1,3,5,6,7,8,9,10
	E	END PDU	None
	F	ENDAK PDU	3,5,6,7,8,9,10
	G	POLL PDU	1,3,6,9
	H	STAT PDU	1,3,6,8,9
	I	USTAT PDU	1,3,6,8,9
	J	RS PDU	1,3,6,7,8,9
	K	RSAK PDU	1,3,6,7,8,9
	L	ER PDU	1,3,6,7,8,9
	M	ERAK PDU	1,3,6,9
Unsuccessful retransmission	O	VT(CC) >= MaxCC	2,4,5,7
	P	timer-no-response expiry	10
Other list elements Error Type	Q	SD or POLL.N(S) error	10
	R	STAT.N(PS) error	10
	S	STAT.N(R) or list elements error	10
	T	USTAT.N(R) or list elements error	10
	U	PDU length violation	ALL
SD Loss	V	SD PDU must be retransmitted	10
Credit Condition	W	Lack of credit	10
	X	Credit obtained	10

Appendix B

The protocol data units for the SSCOP (PDUs) are listed below. The PDU type is coded as a 4-bit field in the PDU header. The table is taken from table 2/Q.2110 in [1].

Table B.1: SSCOP PDU Names and Definitions

Functionality	PDU Name	Code	Description
Establishment	BGN BGAK BGREJ	0001 0010 0111	Request initialization Request Acknowledgement Connection Reject
Release	END ENDAK	0011 0100	Disconnect Command Disconnect Acknowledgment
Resynchronization	RS RSAK	0101 0110	Resynchronization Command Resynchronization Acknowledgment
Recovery	ER ERAK	1001 1111	Recovery Command Recovery Acknowledgemt
Assured Data Transfer	SD POLL STAT USTAT	1000 1010 1011 1100	Seqenced Data delivery State Information Request State Information Response Receiver State Information
Unacknowledged Data Transfer	UD MD	1101 1110	Unnumbered User Data Unnumbered Management Data

Appendix C

Figures C.1 to C.14 illustrate the formats of the SSCOP PDUs (taken from Q.2110) [1]. These figures correspond to the PDUs defined in section 2.6. The features of these formats are noted as following:

- Padding (PAD) field. There will be zero to three unused octets for this field. They are strictly used as filler octets and convey no information.

- Pad Length (PL) field is used to indicate the number of PAD octets present in the PDU.

- PDU length. The maximum information length of a PDU is either K octets (k = 65528) or J octets (j = 65524). These values are reached upon bi-lateral agreements and may specified by SSCF recommendation.

- Reserved field (R or Reserved) are used in each of the PDU. One function of the reserved field is to achieve 32-bit alignment. Other functions are for further study.

- PDU types (indicated as Type) are 4-bit entities that identify the type of the PDU being used (see Appendix B).

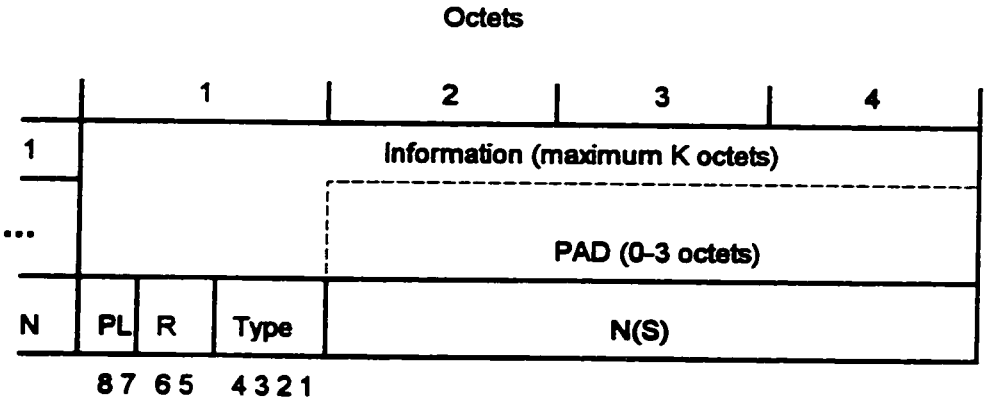


Figure C.1: Sequenced Data PDU (SD PDU)

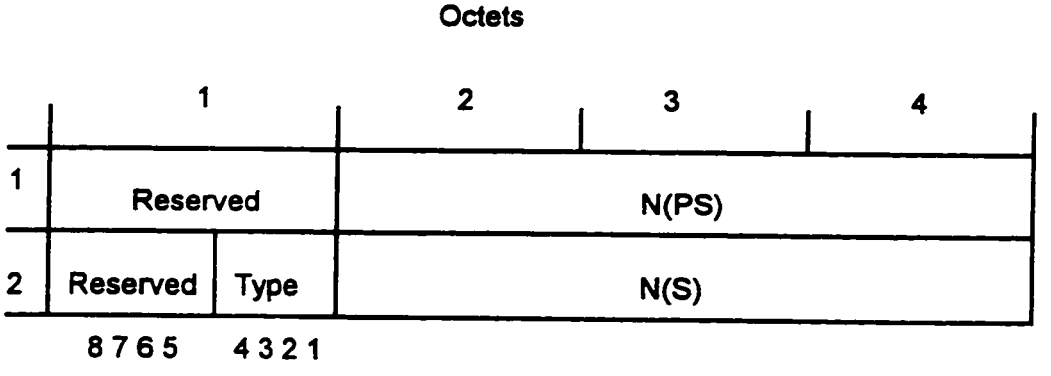


Figure C.2: Poll PDU (POLL PDU)

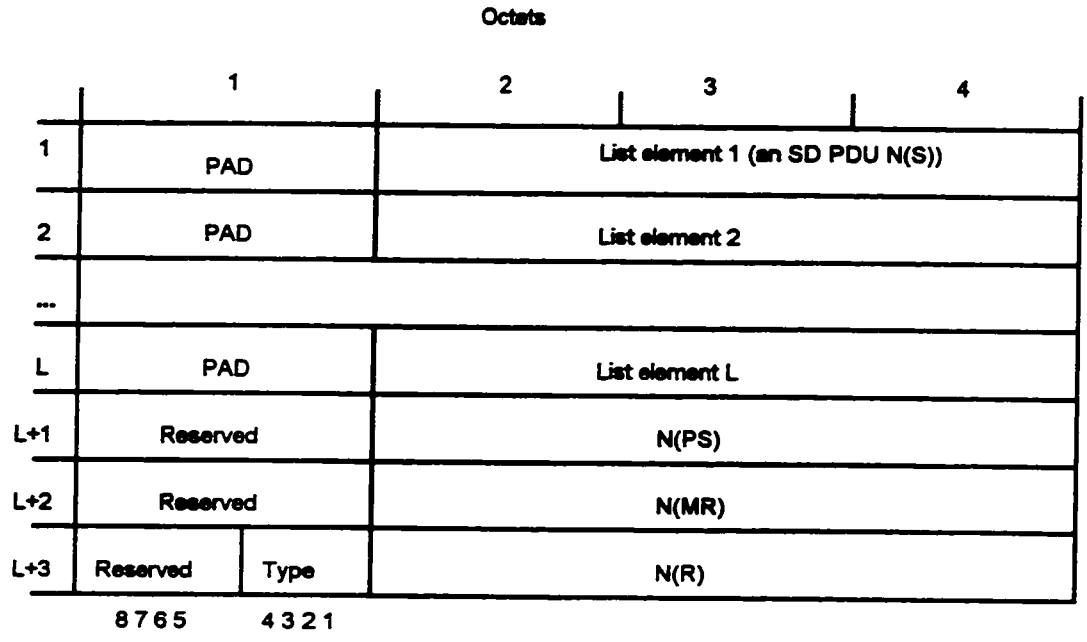


Figure C.3: Solicited Status PDU (STAT PDU)

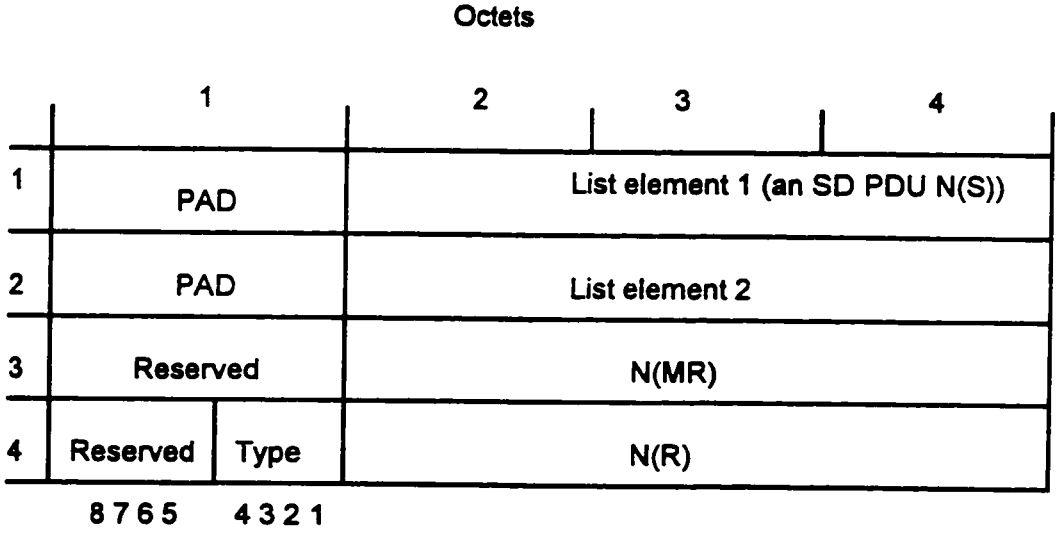


Figure C.4: Unsolicited Status PDU (USTAT PDU)

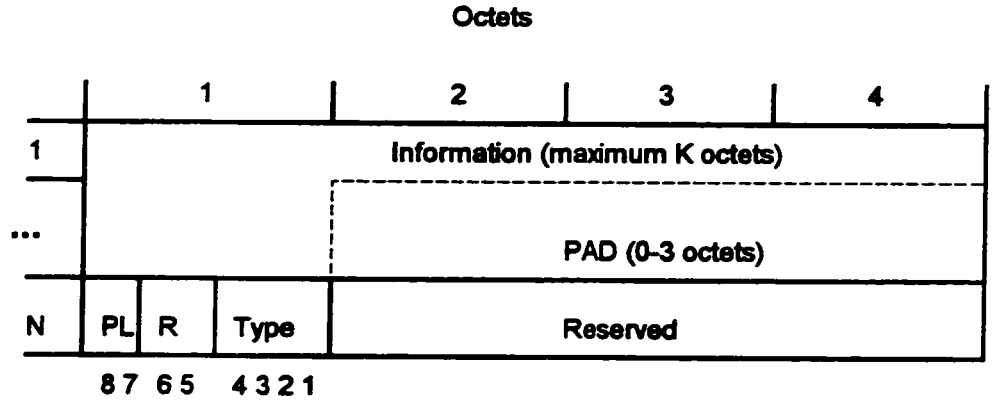


Figure C.5: Unit Data or Management Data PDUs (UD PDU, MD PDU)

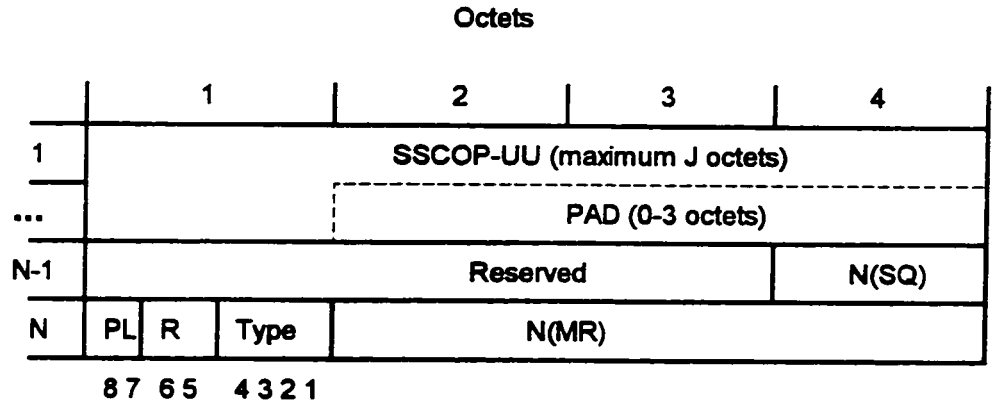


Figure C.6: Begin PDU (BGN PDU)

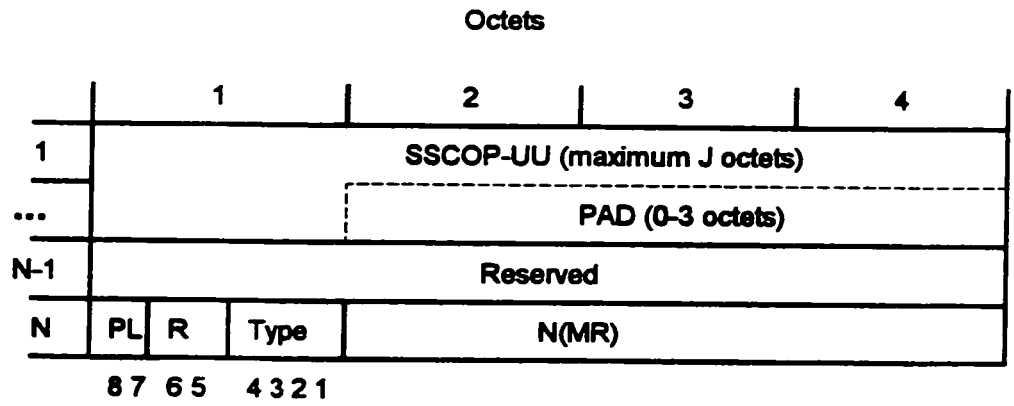


Figure C.7: Begin Acknowledge PDU (BGAK PDU)

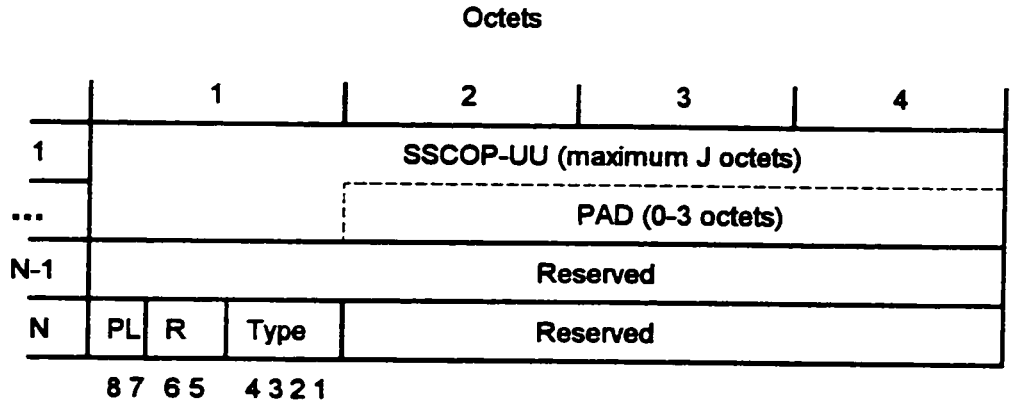


Figure C.8: Begin Reject PDU (BGREJ PDU)

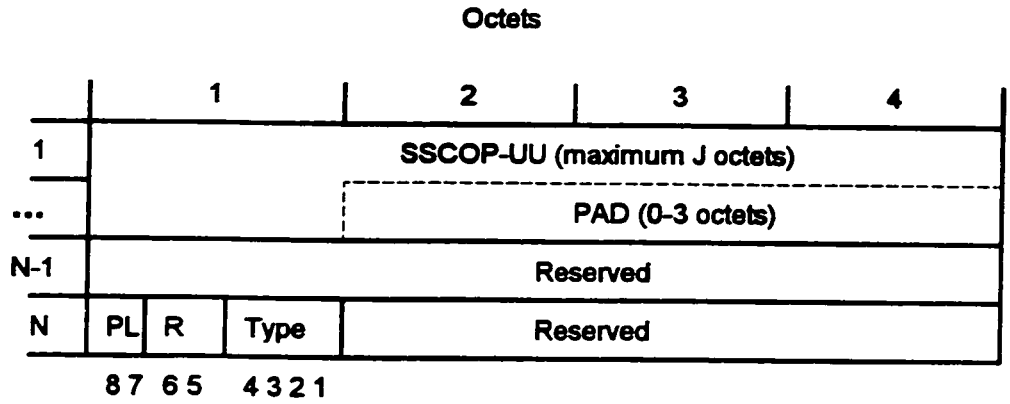


Figure C.9: End PDU (END PDU)

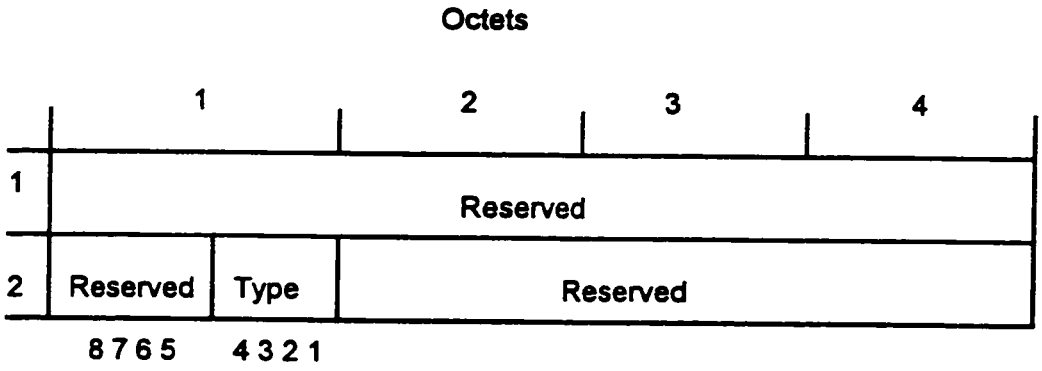


Figure C.10: End Acknowledge PDU (ENDAK PDU)

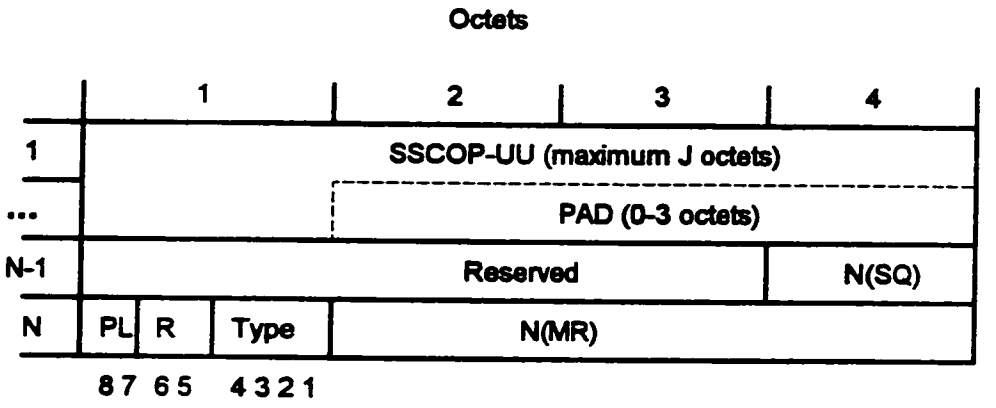


Figure C.11: Resynchronization PDU (RS PDU)

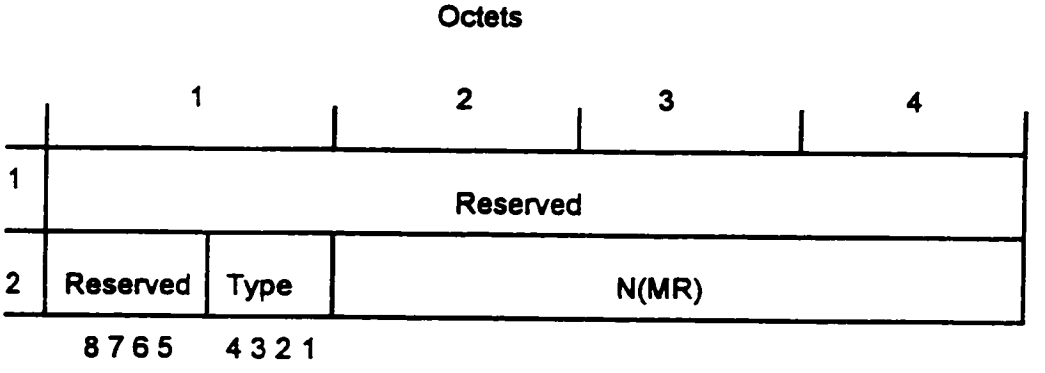


Figure C.12: Resynchronization Acknowledge PDU (RSAK PDU)

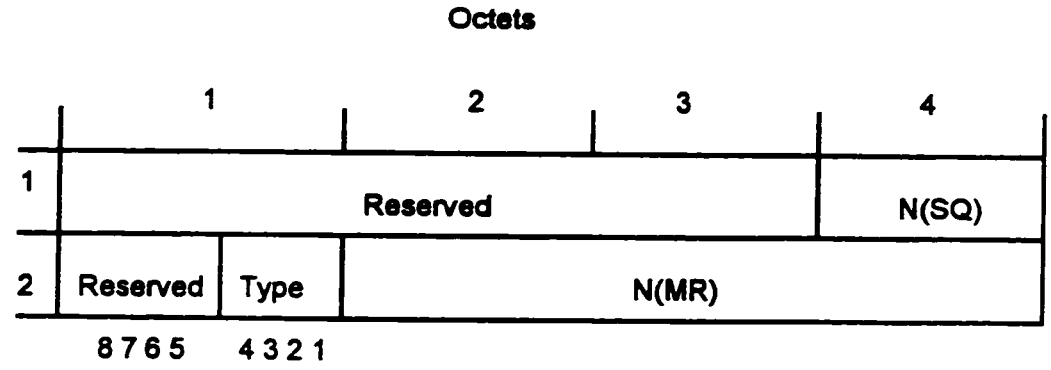


Figure C.13: Error Recovery PDU (ER PDU)

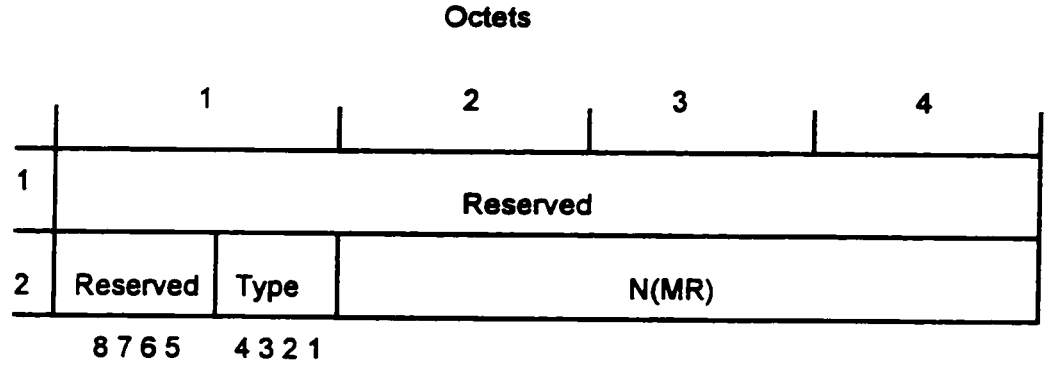


Figure C.14: Error Recovery Acknowledge PDU (ERAK PDU)

Appendix D

In this section as an example we provide the reader with a complete specification for a simple "stop-and-wait" protocol in Estelle. We also give a model for this protocol (figure D.1). In this model two user modules are connected to a simulated network module. Our example is simple and assumes that one user plays the sender and the other plays the receiver.

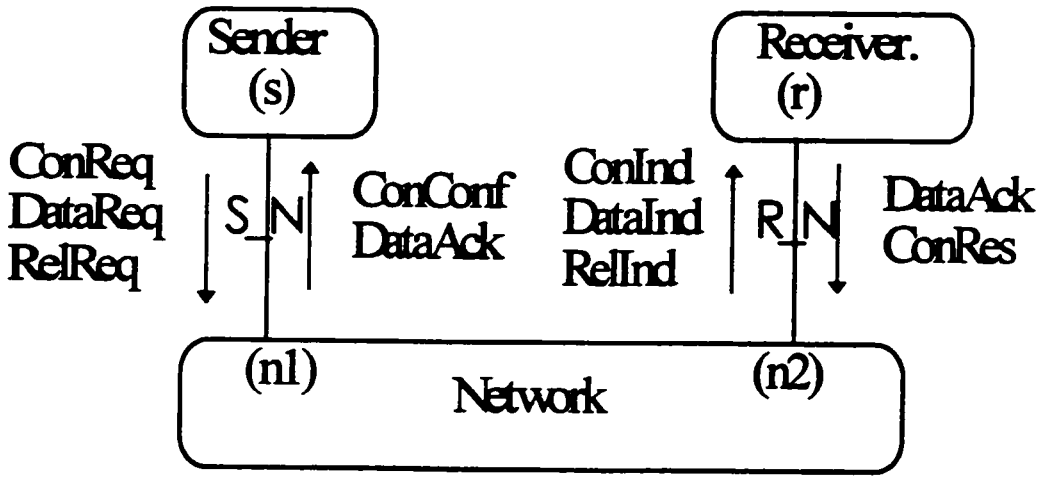


Figure D.1: Stop-and-wait Simulation Model

Initially the sender module is in idle state and starts by requesting a connection. When the connection is acknowledged, the module enters the establish state. In the establish state, the sender module sends a packet and waits for acknowledgment within a fixed period of time. If the acknowledgment did not return in time, the timer transition expires and resents the packet once again.

The network module is responsible for delivering the packet from the sender module to the receiver, and forwarding the acknowledgments from the receiver to the sender.

The receiver module is responsible for responding to a connection request, and produce the acknowledgment for the received packets.

```
{
 this is the specification for the stop-and-wait protocol.
 upon the sending of a packet, the sender will wait for the
 acknowledgment and it also activate a timer. If the ack.
 didn't arrive, the timer will transmit the previous packet
 again.
}
specification altern;
default individual queue;
timescale seconds;


CONST
   maxdelay = 50;{ maximum time to wait for an ack }
   delayvalue = 10;  { used to increment the clock }
   delay_req = 5;
   maxdata = 5;  { maximum number of data to send }

TYPE
   user_data_type = integer;

{ channel definition between the sender and the
            network }


CHANNEL S_N(send,network);
by send:
```

```
        conreq;
        datareq(dataitem:user_data_type);
        relreq;

 by network:
        conconf;
        dataack;

 { channel definition between the recv. and the
              network }

 CHANNEL R_N(recv,network);
 by recv:
        dataack;
        conres;

 by network:
        conind;
        dataind(dataitem:user_data_type);
        relind;

 {================================================}
 PROCEDURE WRITELN_INT(value : integer);primitive;
 PROCEDURE WRITE_INT(value : integer);  primitive;
 PROCEDURE WRITE_CHAR(value : char);    primitive;
 {================================================}

 module sender_h systemprocess; { module headers}
    ip S:  S_N(send);
 end;

 module recv_h systemprocess;
    ip R: R_N(recv);
 end;

 module network_h systemprocess;
    ip n1: S_N(network);
       n2: R_N(network);
 end;

 {================================================}
 BODY send_b for sender_h;
   STATE     idle,establish;


 VAR
        dataitem : user_data_type;
        timer_c : integer;
        Noftry : integer;
        datasent : boolean;

 function getnewitem(item: user_data_type)
                            :user_data_type;
```

```
begin
  getnewitem := item + 1;
end;


   Initialize TO idle
   begin
       Noftry := 0;
       timer_c := 0;
       datasent := false;
   end;
{ transitions for the sender module }
TRANS

   FROM idle
   delay (delay_req)
   NAME requesting : begin
        output s.conreq;
   end;

   FROM idle to establish
   WHEN  s.conconf
   NAME connected : begin
        dataitem := getnewitem(dataitem);
        output s.datareq(dataitem);
        WRITE_INT(dataitem); WRITE_CHAR(' ');
        timer_c := 0;
        datasent := true;
        Noftry := 1;
   end;

   FROM establish
   WHEN s.dataack
   PROVIDED Noftry < maxdata
   NAME newdata : begin
        dataitem := getnewitem(dataitem);
        output s.datareq(dataitem);
        WRITE_INT(dataitem); WRITE_CHAR(' ');
        timer_c := 0;
        datasent := true;
        Noftry := Noftry + 1;
   end;

   FROM establish
   PROVIDED (datasent)
   delay (delayvalue)
   NAME timer : begin
        timer_c := timer_c + 10;
   end;

   FROM establish
   PROVIDED (timer_c > maxdelay) AND (datasent)
   NAME retrans : begin
```

```
          output s.datareq(dataitem);
          WRITE_INT(dataitem); WRITE_CHAR(' ');
          timer_c := 0;
     end;

     FROM establish TO idle
     WHEN s.dataack
     PROVIDED Noftry >= maxdata
     NAME release_req : begin
          output s.relreq;
          datasent := false;
          dataitem := 0;
          Noftry := 0;
          timer_c := 0;
     end;

end; { user body }

{================================================}

Body recv_b for recv_h;


STATE idle,establish;

    initialize To idle
    begin
    end;

TRANS     { transitions for the recv. module }

    FROM idle to establish
    WHEN r.conind
    NAME recv_connect : begin
         output r.conres;
    end;

    FROM establish
    WHEN r.dataind(dataitem)
    NAME recv_data : begin
         WRITELN_INT(dataitem);
         output r.dataack;
    end;

    FROM establish To idle
    WHEN r.relind
    NAME disconnected  : begin
    end;

end; { recv body}

{================================================}
```

```
BODY net_b for network_h;

    initialize
    begin
    end;
{ Network transitions for the sender side }
TRANS

    WHEN n1.conreq
    NAME send_conreq : begin
         output n2.conind;
    end;

    WHEN n1.datareq(dataitem)
    NAME send_data : begin
         output n2.dataind(dataitem);
    end;

    WHEN n1.datareq(dataitem)
    NAME ignore_data : begin
    end;

    WHEN n1.relreq
    NAME send_rel : begin
         output n2.relind;
    end;

{ Network transitions for the receiver side  }
    WHEN n2.dataack
    NAME send_ack : begin
         output n1.dataack;
    end;

    WHEN n2.conres
    NAME send_conf : begin
         output n1.conconf;
    end;

end; { network body}

{================================================}
{ body of the specification module }
MODVAR  { for spec. }

   sender   : sender_h;
   receiver : recv_h;
   net      : network_h;



   INITIALIZE
   begin
        INIT sender WITH send_b;
```

```
        INIT receiver WITH recv_b;
        INIT net WITH net_b;

        CONNECT sender.s TO net.n1;
        CONNECT receiver.r TO net.n2;
   end;
end. { end for spec }
```

Appendix E

```
MODULE user_m SYSTEMPROCESS(fptr1,fptr2: integer);
    IP
      ipXsscopa : sscfXsscop_assur (role_sscf);
      ipXsscopo : sscfXsscop_other (role_sscf);
END;


BODY user_b FOR user_m;

{================================================}
{==== CONSTANT / TYPE / VARIABLE DEFINITIONS ====}

CONST
 {intervals between sending of various types of signals}
    timermin_data_sd = 30;   {ANY integer}
    timermax_data_sd = 35;   {ANY integer}
    timermin_udata = 60;     {ANY integer}
    timermax_udata = 80;     {ANY integer}
    timermin_con_sd = 8;     {ANY integer}
    timermax_con_sd = 20;    {ANY integer}

  VAR
    dataitem : data_t;
    infoitem : info_t;
    release_request : BOOLEAN;
{================================================}
{==============STATE MACHINE PART ==============}

  STATE
      IDLE,ESTABLISH,DISCON;
      { IDLE = idle
        ESTABLISH = connection establish }

  STATESET
    ASTATE = [ESTABLISH,IDLE];


  INITIALIZE
    TO IDLE
    BEGIN
        release_request := false;
    END;
```

```
{      ******TRANSMITTER TRANSITIONS ******      }

  TRANS

  FROM ASTATE

    DELAY (timermin_udata,timermax_udata)
    NAME udata_sending : BEGIN
        getnewdata(fptr1,dataitem);
        OUTPUT ipXsscopo.aa_unitdata_req(dataitem);
    END;


  FROM IDLE

    DELAY (timermin_con_sd,timermax_con_sd)
    NAME est_req_sending : BEGIN
        getnewinfo(fptr2,infoitem);
   OUTPUT ipXsscopo.aa_establish_req(infoitem,true);
    END;


  FROM ESTABLISH

    DELAY (timermin_data_sd,timermax_data_sd)
    NAME data_sd_sending : BEGIN
         getnewdata(fptr1,dataitem);
         IF dataitem.length = 0 THEN
              release_request := true
         ELSE BEGIN
              write_intval(fptr1); }
              write_intval(dataitem.length); }
              OUTPUT ipXsscopa.aa_data_req(dataitem);
         END;
    END;


  FROM ESTABLISH TO DISCON

    PROVIDED release_request
    NAME rel_req_sending : BEGIN
    END;
  FROM DISCON
  DELAY (100)
  NAME just_waitng : begin end;

  FROM DISCON TO IDLE
  DELAY (1300)
  NAME release_connection : BEGIN
        getnewinfo(fptr2,infoitem);
        release_request := false;
        OUTPUT ipXsscopo.aa_release_req(infoitem);
```

```
  END;

{       ****** RECEIVER TRANSITIONS ******        }

  TRANS

  FROM ASTATE
    WHEN ipXsscopo.aa_unitdata_ind(mu)
    NAME udata_reception : BEGIN
       print_data('Module:user_b,
        Trans:aa_unitdata_ind',mu);
    END;

FROM IDLE

    WHEN ipXsscopo.aa_establish_ind (sscop_uu)
    NAME est_ind_treatment_rej: BEGIN
        getnewinfo(fptr2,infoitem);
        OUTPUT ipXsscopo.aa_release_req(infoitem);
    END;


  FROM IDLE TO ESTABLISH

    WHEN ipXsscopo.aa_establish_ind (sscop_uu)
    NAME est_ind_treatment_ack : BEGIN
       getnewinfo(fptr2,infoitem);
       OUTPUT ipXsscopo.aa_establish_res(infoitem,false);
    END;

    WHEN ipXsscopo.aa_establish_con (sscop_uu)
    NAME est_con_treatment : BEGIN
        print_message('connection is established');
     END;

  FROM ESTABLISH,DISCON

    WHEN ipXsscopa.aa_data_ind (mu,seq_no)
    NAME data_reception : BEGIN
   print_a_data('Module:user_b, Trans:aa_data_ind
    ',mu,seq_no);
    END;

    WHEN ipXsscopo.aa_retrieve_ind(mu)
    NAME treat_retrieved_data : BEGIN
   print_data('Module:user_b, Trans:aa_retrieve
     ',mu);
    END;

    WHEN ipXsscopo.aa_retrieve_complete_ind
    NAME retrieval_finished : BEGIN
      {do something important, oh yes}
```

```
    END;

    WHEN ipXsscopo.aa_establish_ind (sscop_uu)
    NAME est_ind_treatment2_ack : BEGIN
       getnewinfo(fptr2,infoitem);
       OUTPUT ipXsscopo.aa_establish_res(infoitem,false);
    END;
    WHEN ipXsscopo.aa_recover_ind
    NAME recover_ind_treatment : BEGIN
       print_message('error recovery is called  ');
       OUTPUT ipXsscopo.aa_recover_res;
    END;

  FROM ESTABLISH TO IDLE

    WHEN ipXsscopo.aa_release_ind(sscop_uu,source)
    NAME rel_ind_treatment : BEGIN
     END;

    WHEN ipXsscopo.aa_release_con
    NAME rel_con_treatment : BEGIN
     END;

{ discarded messages }

  FROM IDLE

    WHEN ipXsscopa.aa_data_ind (mu,seq_no)
    NAME discard_data : BEGIN
          END;

    WHEN ipXsscopo.aa_retrieve_ind(mu)
    NAME discard_retrieved_data : BEGIN
          END;

    WHEN ipXsscopo.aa_retrieve_complete_ind
    NAME discard_retrieval_finished : BEGIN
          END;

    WHEN ipXsscopo.aa_release_ind(sscop_uu,source)
    NAME discard_rel_ind : BEGIN
         END;

    WHEN ipXsscopo.aa_release_con
    NAME discard_rel_con : BEGIN
         END;


  FROM ESTABLISH

    WHEN ipXsscopo.aa_establish_con (sscop_uu)
    NAME discard_est_con : BEGIN
     END;
```

```
  FROM DISCON

    WHEN ipXsscopa.aa_data_ind (mu,seq_no)
    NAME data_reception_ind : BEGIN
  print_a_data('Module:user_b, Trans:aa_data_ind
    ',mu,seq_no);
    END;


END; {user_b}
```

Appendix F

```
MODULE manager_m SYSTEMPROCESS(fptr1 : integer);
    IP
      ipXsscop : manXsscop (role_man);
END;


BODY manager_b FOR manager_m;

{================================================}
{==== CONSTANT / TYPE / VARIABLE DEFINITIONS ===}

  CONST
  {interval between sending of management data units}
    timermin_data_md = 80;
    timermax_data_md = 100;

  VAR
    dataitem : data_t;

{================================================}
{============= STATE MACHINE PART ==============}

  INITIALIZE
    BEGIN
        print_message('Initializing the manager module');
    END;

  TRANS

    WHEN ipXsscop.maa_error_ind(code,count)
    NAME sscop_error : BEGIN
   print_error('Module: manager_b, Trans:maa_error_ind
    ',code,count);
    END;

    WHEN ipXsscop.maa_unitdata_ind(mu)
    NAME man_data_rec : BEGIN
     print_data('Module:manager_b,
     Trans:maa_unitdata',mu);
    END;

  TRANS
```

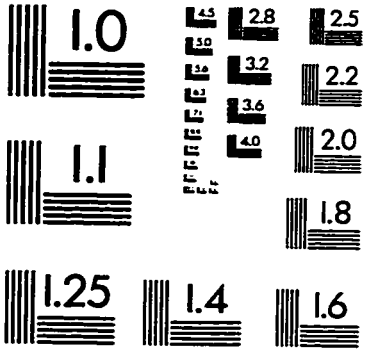
```
        DELAY (timermin_data_md,timermax_data_md)
        NAME data_md_sending : BEGIN
            getnewdata(fptr1,dataitem);
            print_message('sending data from the file:');
            write_intval(fptr1);
            OUTPUT ipXsscop.maa_unitdata_req(dataitem);
        END;
END; {manager_b}
```

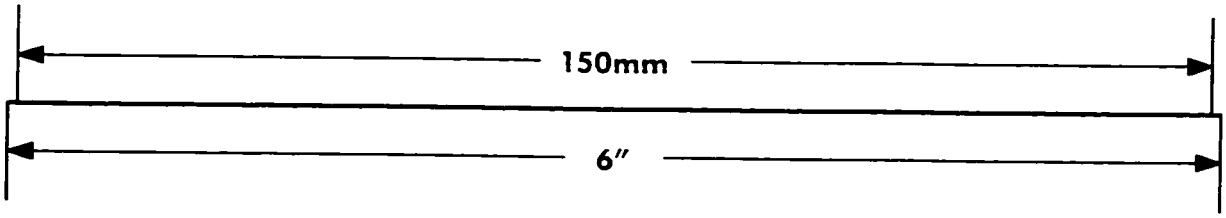
Appendix G

Table G.1: Simulation Results

Senarios	Lossrate-values	Results
Clean Environment	0	No error reported by any observer
Hostile Environment: 1. Data packet loss	2	No error reported by the observer at the user module. The observer at the Management module reported an error for every missing data packet.
2. Control packet loss	3	As long as the timer allowed, the SSCOP waited for the next control packet transmission.
	10	The connection is aborted when the timer no-response expired.
3. All type of packets loss	2	Similar to senario 2.
More hostile environment by tampering with the SSCOP values		The error recovery procedures was called as the SSCOP moved from the state 10 to state 7.

TEST TARGET (QA-3)





APPLIED IMAGE, Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989