OSIRIDE Session Layer

CSIRIDE Session Layer

Fausto Caneschi
Marina Bon
Silvio Comel

Report CNUCE C84-8

Istituto CNUCE
PISA
May 1984

OSIRIDE Session Layer

Implementation Specifications for the OSIRIDE Session Layer

Fausto Caneschi[1]
Marina Bon[2]
Silvio Comel[3]

[1]CNUCE - Institute of C.N.R. - via S. Maria 36 - Pisa
[2]Ing. C. OLIVETTI & C. via Jervis 77 - Ivrea (TO)
[3]ITALSIEL via Isonzo 21/b - Roma

Doc. **OSIRIDE/83/SES/02**

January 1984

INTRODUCTION

This document gives a detailed description of the OSIRIDE Session Layer, which has been generally described in [1].

The document is structured in two parts, following the guidelines for the production of the OSIRIDE final documents described in [2], which are:

1. A first part, in which the services and the protocol of the OSIRIDE Session Layer are described by words, with the same style as it can be found in the Standard documents [3] and [4], from which it derives. This part is mainly taken from the already mentioned and related OSIRIDE document [1].

2. A second part, which gives the detailed description of the implementation specifications for the OSIRIDE Session Layer. Main characteristics of this second part are:

   - Internal Layer's structure
   - Data structures
   - Upper level interface (including the services offered to the Local Management System).
   - Lower level interface
   - Internal scheduler description

TABLE OF CONTENTS

LIST OF FIGURES

OSIRIDE Session Layer

First Part

1.0 INTRODUCTION

The first OSIRIDE implementation for the layer 5 provides for the Session Layer protocol defined by ISC in [3] and [4]

The ISO Session protocol defines a number of functional units, which may be negotiated.

In this chapter the choices which have been done for the OSIRIDE Session layer will be described, following the same style as the related ISO Draft Proposals [3] and [4].

1.1 SCOPE

This part defines in an abstract way both the externally visible service provided by the OSIRIDE Session Layer and the internal OSIRIDE Session Protocol mechanisms.

1.2 DEFINITIONS

Apart from the terms defined in the Reference Model [5], the following are Session specific terms and are defined in this section.

1.2.1 Session Service Definitions

Session Services User; SS User An abstract representation of the totality of those entities within a single system that make use of the Session Service.

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

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

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

sending SS user An SS User that acts as a source of data during the data transfer phase of a session-connection.

receiving SS User An SS user that acts as a sink of data during the data transfer phase of a session-connection.

Note: An SS User can be both sending and receiving SS User simultaneously.

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

acceptor; accepting SS User An SS User that accepts a particular action.

token An attribute of a session-connection which is dynamically assigned to one SS user at a time to permit certain services to be invoked.

1.2.2 Session Protocol Definitions

transport service provider An abstract machine which models the totality of the entities providing the transport service, as viewed by a session entity.

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

sending session entity A session entity that sends a given SPDU.

receiving session entity A session entity that receives a given SPDU.

owner (of a token) The session entity to whom a token is assigned.

proposed parameter The value for a parameter that the initiator indicates in a CONNECT SPDU that it wishes to use over the Session connection.

selected parameter The value for a parameter that the responder indicates in an ACCEPT SPDU that it has chosen for user over the Session connection.

invalid SPDU An SPDU which does not comply with the requirements of the OSIRIDE Session protocol for structure and encoding.

protocol error An SPDU whose use does not comply with the procedures agreed for this session-connection.

transparent (data) SS user data which is transferred intact between session entities, and which is unavailable for use by the session entities.

command identifier Heading information that identifies the command SPDU concerned.

response identifier Heading information that identifies the response SPDU concerned.

SPDU code a command identifier or a response identifier.

length indicator An indicator which represents the length in octets of an associated field or group of fields.

parameter identifier; parameter group identifier An identifier that indicates the type of information contained in an associated field or group of fields.

Note: Both terms are defined and used in order to maintain compatibility with CCITT recommendation S.62, where a parameter group identifier is defined as "a special case of a parameter identifier which indicates that the associated field consists entirely of a group of parameters, each identified by a parameter identifier".

parameter value This value is the information that represents the value of the parameter identified by either a PI or a PGI.

field A group of one or more bits within a single octet or a group of one or more octets, used to represent a particular set of information.

Session Protocol Machine; SPM The instances of communication of this protocol are expressed in terms of the operations performed by an abstract machine, the Session Protocol Machine.

V(M) An abstract local variable which is used in order to allow the peer entities to maintain the coherence of the serial numbers. V(M) contains the the serial number of the next synchronization point to be requested by the SS user.

V(A) An abstract local variable which is used in order to maintain the lowest serial number to which a synchronization point confirmation is expected. No confirmation is expected when V(A) = V(M).

V(R) An abstract local variable which is used in order to maintain the lowest serial number to which resynchronization is permitted.

V-SC An abstract local variable used to determine whether or not the SS user has the right to send minor mark confirmations. The variable has the following values:

V-SC = true: the SS user has the rigth to issue minor mark confirmations.

V-SC = false: the SS user does not have the rigth to issue minor mark confirmations.

V-ACT Is an abstract local variable which is used in order to determine the availability of the activity management service, and to detect if an activity is in progress. In the first OSIRIDE version, activity is not supported, but the variable has been maintained for compatibility.

1.3 ABBREVIATIONS

1.3.1 Session Service Abbreviations

BS	basic synchronized subset
SS	session service.
SC	session connection.
SSAP	session service access point.
SSDU	session service data unit.
XSSDU	expedited session service data unit.

1.3.2 Sessiom Protocol Abbreviations

SPDU	session protocol data unit
TSDU	transport service data unit
SI	SPDU identifier
LI	length indicator
PI	parameter identifier
PGI	parameter group identifier
PV	parameter value
TAB	abort timer

OSIRIDE Session Layer

2.0 SESSION SERVICE

OSIRIDE Version 1 implements the following Functional Units[1]:

- **Kernel**
- **Half Duplex**
- **Full Duplex**
- **Negotiated Release**
- **Minor Synchronize**
- **Major Synchronize**
- **Resynchronize**
- **Typed data**

In the following sections, the session primitives contained in the above mentioned Functional Units will be described in terms of the supported parameters.

---

1 For the definition of the term Functional Unit, see [3].

2.1 KERNEL FUNCTIONAL UNIT

The Kernel Functional Unit contains the following session primitives:

- S_CONNECT
- S_DATA
- S_RELEASE
- S_U_ABORT
- S_P_ABORT

2.1.1 S_CONNECT

Fig. 1 describes the S_CONNECT related parameters.

Parameter	Request	Indicat.	Response	Confirm
Identifier	X	X	X	X
Calling SSAP id	X	X(=)	-	-
Called SSAP id	X	X(=)	X(=)	X(=)
Result	-	-	X(1)	X(=)
Qual. of service	X	X	X	X
Session requir.	X(2)	X(=)	X(9)	X(=)
Serial number	X(3)	X(=)	X(4)	X(=)
Data token(5)	X(6)	X(=)	X(7)	X(=)
Release token(5)	X(6)	X(=)	X(7)	X(=)
Sync-min token(5)	X(6)	X(=)	X(7)	X(=)
Major token(5)	X(6)	X(=)	X(7)	X(=)
User data	X(8)	X(=)	X(8)	X(=)

Fig. 1. S_CONNECT parameters

Notes:

1. The value of the result parameter can be one of:

   A. accept

   B. reject (user)

   C. reject (provider + reason of failure)

Only the first two values can be present in a response.

2. The following session requirements are indicated[2]:

   A. half-duplex (value := **user**)

   B. duplex (value := **user**)

   C. exceptions (value := **no**)

   D. typed data (value := **user**)

   E. negotiated release (value := **user**)

   F. synchronize-minor (value := **user**)

   G. synchronize-major (value := **user**)

   H. resynchronization (value := **user**)

   I. session expedited (value := **no**)

   J. activity management (value := **no**)

   K. capability data (value := **no**)

3. This parameter is only present when minor-sync token or a major token is available.

4. Value in response/confirmation takes precedence over value in request/indication.

5. This parameter is required only if the token is available.

6. Value may be one of:

   A. initiator side

   B. acceptor side

---

[2] In the first OSIRIDE implementation, the session requirements are forced in such a way as to support the Basic Synchronized Subset. For each session requirement, the value may be:

no: means that the corresponding Functional Unit is not supported.

yes: means that the corresponding Functional Unit is **always** supported by an OSIRIDE implementation.

user: means that value of **yes** or **no** has to be provided by the user. If nothing is specified, **no** is assumed by default.

C. acceptor chooses

7. Value may be one of:

   A. initiator side

   B. acceptor side

   If the value in the request/indication is not "acceptor chooses", the parameter is absent.

8. User data of size limited to 512 octets.

9. The session requirements specified in the response indicate the accepted session user requirements to the initiator. These requirements have to be selected from those proposed in the indication. The acceptor may not indicate both half-duplex service and duplex service in the response.

In OSIRIDE, the Quality of Service parameters are selected according to the following criterion:

1. Some QoS parameters are not selectable, because they are not implemented:

   - session connection protection
   - session connection priority
   - transit delay
   - session connection establishment delay
   - session connection establishment failure probability
   - transfer failure probability
   - session connection release delay
   - session connection release failure probability

2. Some QoS parameters are **always** selected by the OSIRIDE Session Layer, and are not selectable by the Session user:

   - optimized dialogue transfer, which is always set to **yes**;
   - extended control, which is always set to **yes**.

3. Some QoS parameters are selectable by the OSIRIDE Session user, by selecting a particular Session Profile:

   - throughput
   - session connection resilience

2.1.2 S_DATA

Fig. 2 describes the S_DATA related parameters.

Parameter	Request	Indicat.
Data	X(1)	X(=)

Fig. 2. S_DATA parameters

Note:

1. The size of the SSDU is an integral number of octets greater than zero.

2.1.3 S_RELEASE

Fig. 3 describes the S_RELEASE related parameters.

Parameter	Request	Indicat.	Response	Confirm
Result	-	-	X(1)	X(=)
User data	X(2)	X(=)	X(2)	X(=)

Fig. 3. S_RELEASE parameters

Notes:

1. The value of the result parameter can be affirmative or negative. This parameter may take the "negative" value only if the release token is available. This means that the Negotiated Release Functional Unit has been selected.

2. User data limited to 512 octets.

2.1.4 S_U_ABORT

Fig. 4 pag. 12 describes the S_U_ABORT related parameters.

Parameter	Request	Indicat.
User data	X(1)	X(=)

Fig. 4. S_U_ABCRT parameters

Note:

1. User data limited to 9 octets.

2.1.5 S_P_ABORT

Fig. 5 describes the S_P_ABORT related parameters.

Parameter	Indicat.
Reason	X(1)

Fig. 5. S_U_ABORT parameters

Note:

1. Reason code values may be one of:

   A. protocol error

   B. undefined

2.2 TYPED DATA FUNCTIONAL UNIT

The Typed Data Functional Unit contains the following session primitive:

- **S_TYPED_DATA**

2.2.1 S_TYPED_DATA

Fig. 6 describes the S_TYPED_DATA related parameters.

Parameter	Request	Indicat.
Data	X(1)	X(=)

Fig. 6. S_TYPED_DATA parameters

Note:

1. The size is an integral number of octets greater than zero.

2.3 HALF DUPLEX FUNCTIONAL UNIT

The Half Duplex Functional Unit contains the following primitives:

- **S_TOKEN_GIVE**
- **S_TOKEN_PLEASE**

The two above quoted primitives are also used in all those Functional Units which need a Token management mechanism, and are listed in Fig. 7 pag. 14. This means that the S_TOKEN_GIVE and S_TOKEN_PLEASE primitives here described are not only intended for the Data Token management, which is in the scope of the Half Duplex Functional Unit, but are also valid for the other Functional Units which use tokens.

Functional Unit	Token
Negotiated Release	Release token
Half Duplex	Data token
Minor Synchronize	Synchronize minor token
Major Synchronize	Major/activity token

Fig. 7. Functional Units using tokens

2.3.1 S_TOKEN_GIVE

Fig. 8 describes the S_TOKEN_GIVE related parameters.

Parameter	Request	Indicat.
Token	X(1)	X(=)

Fig. 8. S_TCKEN_GIVE parameters

Note:

1. The value is any combination of:
   A. data token
   B. synchronize-minor token
   C. major token
   D. release token

2.3.2 S_TOKEN_PLEASE

Fig. 9 pag. 15 describes the S_TOKEN_PLEASE related parameters.

Parameter	Request	Indicat.
Token	X(1)	X(=)
User data	X(2)	X(=)

Fig. 9. S_TOKEN_PLEASE parameters

Notes:

1. The value is any combination of:
   A. data token
   B. synchronize-minor token
   C. major token
   D. release token

2.6.1 S_RESYNCHRONIZE

Fig. 12 describes the S_RESYNCHRONIZE related parameters.

Parameter	Request	Indicat.	Response	Confirm
Resyn. Type	X(1)	X(=)	-	-
Serial number	X(2)	X(=)	X	X(=)
Tokens	X(3)	X(=)	X(4)	X(=)
User data	X(5)	X(=)	X	X(=)

Fig. 12. S_RESYNCHRONIZE parameters

Notes:

1. Type value defines the Resync option. The value is one of:

   A. restart

   B. set

   C. abandon

   If there is contention between S_RESYNCHRONIZE requests, "abandon" has priority.

2. If the resynchronize type value is "restart", then the synchronization point serial number in the request must be not less than that of the last major synchronization point. The value in the response must not be less than that of the last major synchronization point, and not greater than that in the request.

   If the resynchronize type value is "abandon" then the new synchronization point serial number is set by the SS provider, and it will be greater than any preceding value for this session connection.

   If the resynchronize type value is "set" then the new synchronization point serial number is set by the SS user. The Session Service does not perform any validation on the synchronization point, as in the case of "abandon".

3. For the tokens which are not available in the session connection, the value is "not specified". For the tokens which are available in the session connection, the values may be:

   A. the token is assigned to the initiator

   B. the token is assigned to the acceptor

   C. the acceptor chooses.

4. If the value in the request/indication is "acceptor chooses the side", then the acceptor makes his own choice for positioning the token:

   A. the token is assigned to the initiator

   B. the token is assigned to the acceptor

   For all other values of the request/indication, the response/confirmation contains the same as the request/indication.

5. User data 512 octets maximum.

6. In case of collision of Resynchronization requests, user data may be lost. The collision cases are solved according the table in Fig. 13 pag. 19.

User is waiting for	User receives			
	RR	RS	RA	AB
S_RESYNC(restart) confirm	X	X	X	X
S_RESYNC(set) confirm		X	X	X
S_RESYNC(abandon) confirm			X	X

Key:

AB: S_P_ABORT indication or
S_U_ABORT indication or

RA: S_RESYNCHRONIZE(abandon) indication

RR: S_RESYNCHRONIZE(restart) indication

RS: S_RESYNCHRONIZE(set) indication

Fig. 13. Indications resulting from collision resolution

3.0 SESSION PROTOCOL

This clause is based on the ISO Session Protocol [4], with the simplifications which have been decided for OSIRIDE.

3.1 SERVICES ASSUMED FROM THE TRANSPORT LAYER

The OSIRIDE Session protocol assumes the use of the Transport service defined in DP 8072 [9], which is detailed in [10].

Information is transferred to and from the TS provider in the transport service primitives listed in Fig. 14 pag. 20.

Primitive	X/Y	Parameters
T_CONNECT request indicat	X	To Transport address, from transport address, Expedited data option, Quality of service, TS user data
T_CONNECT resp.	X	Responding address, Quality of service, Expedited data option, TS user data
T_DATA request indic.	X	TS user data
T_EXPEDITED_DATA request indic.	Y	TS user data
T_DISCONNECT request	X	TS user data
T_DISCONNECT indic.	X	reason, TS user data

Fig. 14. Transport service primitives

X: The session protocol assumes this facility as always available.

Y: The session protocol assumes that this facility is provided by the transport layer when requested by the session layer.

Note: In the first OSIRIDE implementation, the T_EXPEDITED_DATA service is **always** assumed as available.

3.2 PROTOCOL FUNCTIONAL UNITS

In the following sections, the functional units and the SPDUs defined in the ISO Session protocol, and the actions which are taken by the session protocol machine are listed.

Fig. 15 pag. 22 summarizes the Functional Units and the related SPDUs which are implemented in the OSIRIDE Session protocol.

Functional Unit	SPDUs	Reference
Kernel	CONNECT	"4.1 CONNECT SPDU"
	ACCEPT	"4.2 ACCEPT SPDU"
	REFUSE	"4.3 REFUSE SPDU"
	FINISH	"4.4 FINISH SPDU"
	DISCONNECT	"4.5 DISCONNECT SPDU"
	ABORT	"4.6 ABORT SPDU"
	ABORT ACCEPT	"4.7 ABORT ACCEPT SPDU"
	DATA TRANSFER	"4.9 DATA TRANSFER SPDU"
Use of half duplex	PLEASE TOKENS	"4.11 PLEASE TOKENS SPDU"
	GIVE TOKENS	"5.3.10 GIVE TOKENS SPDU"
Typed data	TYPED DATA	"4.12 TYPED DATA SPDU"
Negotiated release	NOT FINISHED	"4.8 NOT FINISHED SPDU"
	PLEASE TOKENS	"4.11 PLEASE TOKENS SPDU"
	GIVE TOKENS	"5.3.10 GIVE TOKENS SPDU"
Minor synchronize	MINOR SYNC POINT	"4.13 MINOR SYNCHRONIZATION POINT SPDU"
	MINOR SYNC ACK	"4.14 MINOR SYNC ACK SPDU"
	PLEASE TOKENS	"4.11 PLEASE TOKENS SPDU"
	GIVE TOKENS	"5.3.10 GIVE TOKENS SPDU"
Major synchronize	MAJOR SYNC POINT	"4.15 MAJOR SYNCHRONIZATION POINT SPDU"
	MAJOR SYNC ACK	"4.16 MAJOR SYNC ACK SPDU"
	PLEASE TOKENS	"4.11 PLEASE TOKENS SPDU"
	GIVE TOKENS	"5.3.10 GIVE TOKENS SPDU"
Resynchronize	RESYNCHRONIZE	"4.17 RESYNCHRONIZE SPDU"
	RESYNCHRONIZE ACK	"4.18 RESYNCHRONIZE ACK SPDU"
	PREPARE	"4.19 PREPARE SPDU"

Fig. 15. Functional Units in OSIRIDE Session Layer

3.3 USE OF A TRANSPORT CONNECTION

This clause defines the way that the transport connection is used by the SPM.

3.3.1 Assignement of a session connection to a transport connection.

3.3.1.1 Purpose

Assignement of a session connection to a transport connection.

3.3.1.2 Transport service primitive

This procedure uses the following transport primitives:

T_CONNECT

T_DISCONNECT

3.3.1.3 SPDU used

No SPDUs are used during assignement to a transport connection.

3.3.1.4 Description

Before a session connection can be operated or used it shall be assigned to a transport connection **with a suitable quality of service.** Both SPMs are aware of this assignement. A session connection may be assigned to an existing transport connection suitable for reuse, or a new transport connection may be created for the purpose. **This latter case is the only permitted one in OSIRIDE.**

The **Quality of service** parameter of the transport connection is selected by means of the **User profile.**

In OSIRIDE, when the session connection is terminated, the underlying transport connection is also terminated.

3.4 USE OF TRANSPORT NORMAL DATA

3.4.1 Purpose

To convey SPDUs in the user data field of transport normal data primitive.

3.4.2 Transport service primitives

This procedure uses the following transport primitives:

T_DATA

3.4.2.1 SPDU used

The following SPDUs are sent on the transport normal flow:

CONNECT

ACCEPT

REFUSE

NOT FINISHED

DISCONNECT

DATA

TYPED DATA

PLEASE TOKENS

GIVE TOKENS

MINOR SYNCHRONIZATION POINT

MINOR SYNC ACK

MAJOR SYNCHRONIZATION POINT

MAJOR SYNC ACK

RESYNCHRONIZE

RESYNCHRONIZE ACK

3.4.3 Transfer

3.4.3.1 Segmenting

Each SPDU is mapped one to one into an SPDU, unless segmenting has been selected, in which case an SSDU may be mapped into more than one SPDU for the duration of the session connection.

The TSDU maximum size for each direction is indicated in the CONNECT SPDU. The TSDU may contain any number of octets up to the agreed size limitation. When segmenting is employed, the SPM shall

indicate the beginning and end of an SSDU by setting the end of SSDU bit.

In OSIRIDE version 1, segmenting is not implemented.

3.4.4 Concatenation

Each SPDU is defined in the table of Fig. 16 pag. 26 as belonging to one of the following categories:

A. Category 0 SPDUs which may be mapped one-to-one onto a TSDU or may be concatenated with one or more Category 2 SPDUs.

B. Category 1 SPDUs which are always mapped one-to-one into a TSDU.

C. Category 2 SPDUs which are never mapped one-to-one into a TSDU.

Basic concatenation of a Category 0 SPDU with a single Category 2 SPDU, defined as valid in Fig. 17 pag. 26, may always be mapped into a single TSDU.

If the receiving SPM has indicated that it can accept extended concatenation **(which is the OSIRIDE case)**, the sending SPM may map extended concatenations of a Category 0 SPDU with two or more Category 2 SPDUs defined as valid and in the order indicated in Fig. 18 pag. 26, onto a single TSDU. The valid forms of TSDUs are illustrated in Fig. 19 pag. 27.

Any other concatenation of SPDUs is defined as invalid.

3.4.5 Processing order of concatenated SPDUs

On receipt of SPDUs that have been concatenated using basic concatenation, the Category 2 SPDUs are processed before the Category 0 SPDU.

On receipt, SPDUs that have been concatenated using extended concatenation are processed in the following order:

A. DATA TRANSFER SPDU

B. MINOR SYNCHRONIZATION POINT SPDU or
MINOR SYNC ACK SPDU or
MAJOR SYNCHRONIZATION POINT SPDU or
MAJOR SYNC ACK SPDU

C. GIVE TOKENS SPDU or
PLEASE TOKENS SPDU

Category 0 SPDUs	Category 1SPDUs	Category 2 SPDUs
GIVE TOKENS	CCNNECT	DATA TRANSFER
PLEASE TOKENS	ACCEPT	MINOR SYNC POINT
	REFUSE	MINOR SYNC ACK
	FINISH	MAJOR SYNC POINT
	DISCONNECT	MAJOR SYNC ACK
	NCT FINISHED	RESYNCHRONIZE
	ABORT	RESYNCHRONIZE ACK
	ABORT ACCEPT	
	TYPED DATA	

Fig. 16. Category 0, 1 and 2 SPDUs

First SPDU	Second SPDU
GIVE TOKENS	DATA TRANSFER
PLEASE TOKENS	MINOR SYNC POINT
PLEASE TOKENS	MINOR SYNC ACK
GIVE TOKENS	MAJOR SYNC POINT
PLEASE TOKENS	MAJOR SYNC ACK
GIVE TOKENS*	RESYNCHRONIZE
PLEASE TOKENS	RESYNCHRONIZE ACK

Key: An asterisk indicates that the Token Item parameter is not present in the GIVE TOKENS SPDU, which in this case only acts as an header.

Fig. 17. Valid basic concatenation of SPDUs

First SPDU	Second SPDU	Third SPDU
GIVE TOKENS	MINOR SYNC POINT	DATA TRANSFER
GIVE TOKENS	MINOR SYNC ACK	DATA TRANSFER
GIVE TOKENS	MAJOR SYNC POINT	DATA TRANSFER
GIVE TOKENS	MAJOR SYNC ACK	DATA TRANSFER

Fig. 18. Valid extended concatenations of SPDUs

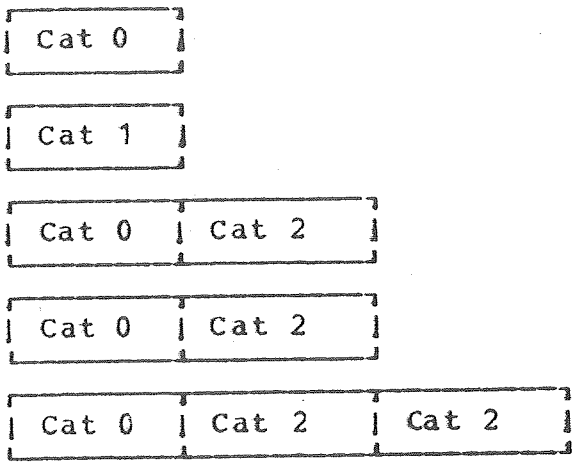


Fig. 19. TSDU structures

3.5 USE OF TRANSPORT EXPEDITED DATA

3.5.1 Purpose

To convey SPDUs in the user data field of transport expedited data. primitive.

3.5.2 Transport service primitives

This procedure uses the following transport primitives:

T_EXPEDITED

3.5.2.1 SPDU used

The following SPDUs are sent on the transport expedited flow:

ABORT

PREPARE

Besides, the ABORT ACCEPT SPDU may be received in the transport expedited flow, although **this SPDU is not implemented in OSIRIDE.**

3.6 USE OF TRANSPORT DISCONNECTION

3.6.1 Purpose

To release a Transport connection.

3.6.2 Transport service primitives

This procedure uses the following transport primitives:

T_DISCONNECT

3.6.3 SPDU used

No SPDUs are used.

3.6.4 Description

After the Session connection has been released or aborted, and the Transport connection is not to be reused, which is the OSIRIDE case, the Transport connection is disconnected.

When a T_DISCONNECT indication is received, as a result of an error detected by the transport service provider, the SPM issues an S_P_ABORT indication to the local SS-user.

When issuing a T_DISCONNECT request, the OSIRIDE implementation **will not user the user data field.**

4.0 ELEMENTS OF PROCEDURE RELATED TO SPDUS

The following specification defines the valid sequences of operation in the protocol. This clause specifies SPM behaviour on receipt of SPDUs when error conditions do not exist.

4.1 CONNECT SPDU

The CONNECT SPDU is transmitted by the initiator of a transport connection in order to initiate a session connection. Content of CONNECT SPDU

A CONNECT SPDU contains:

1. **A Connection identifier parameter group**, which is supplied to identify this specific session connection. This parameter has no effects on the SPM, but may be used for accounting and/or statistic purposes. It is set by the initiator of the session connection, and **it is not changed, but transmitted back** in the ACCEPT message. The Connection identifier parameter group is made of three subfields, that are:

   A. Calling SS-user Reference parameter;

   B. Common Reference parameter;

   C. Additional Reference Information parameter;

   In OSIRIDE, only the Calling SS-user Reference parameter is used, and is structured in the following way:

   <Calling SS-user Reference> ::= <OSIRIDE Host id>
   <Application> <time>

   <OSIRIDE Host id> ::= <binary number> (1 octet)

   Note: The OSIRIDE Host id is a binary number univoquely assigned to each OSIRIDE host when that Host is declared an CSIRIDE host.

   <Application> ::= <characters string>

   Note: The Application is a character string **coded in ASCII** which identifies the initiator of the session connection, as it declared itself to CSIRIDE.

   <time> ::= <year> <month> <day> <hour> <minute> <second>

Note: The time which is coded in the connection identifier is expressed as **a string of 6 octets**, each of them representing the related figure as a binary number (for the year, the last two digits are considered). The Session Addresses associated with the initiator and acceptor are provided by the OSIRIDE Access Method (ISIDE [8]), and are carried on transparently by the Session provider. It is worth to remind here, however, that those addresses are coded in ASCII, and the acceptor address is used by the receiving session entity to identify the user which the connection request is directed to. The initiator address corresponds to the <Application> value of the Connection identifier parameter.

2. **A Connect/Accept parameter group**, which contains:

   A. A protocol Options parameter, **which is set to 1 (one).** This means that the concatenation option is **always** supported.

   B. A Version Number parameter, **which is set to 1 (one).**

   C. A TSDU maximum size parameter, **which is set to 0 (zero).** This means that segmenting is not supported.

   D. In the case where synchronization is required on the session connection, the user may supply an Initial Serial Number which may be negotiated with the acceptor.

   E. Initial token settings will be supplied for each token available on this connection, as derived from the functional unit bits in the accepted Session requirements parameters.

3. **The Session Requirements parameter**, which contains a bitmap of the SS users proposed functional units. The following list represent the mapping between bits in that bitmap and services specified by the SS users in the connect request:

   bit 1 (half-duplex) ==> data token defined

   bit 2 (full-duplex) ==> data token not defined

   bit 3 (expedited) ==> **always set to 0 (zero)**

   bit 4 (minor sync) ==> minor token defined

   bit 5 (major sync) ==> major token defined

   bit 6 (resync) ==> resynchronization token defined

   bit 7 (activity) ==> **always set to 0 (zero)**

   bit 8 (negotiated release) ==> release token defined

   bit 9 (capability data) ==> **always set to 0 (zero)**

Elements of procedure related to SPDUs

bit 10 (exceptions) ==> **always set to 0 (zero)**

bit 11 (typed data) ==> typed data used

4. **Calling SSAP identifier and Called SSAP identifier**, which are assigned by the OSIRIDE Access Method.

5. **User data**, which allows up to 512 octets of user data to be passed from the initiating SS user to the accepting SS user.

4.1.1 Sending the CONNECT SPDU

An S-CONNECT.request results in the assignment of a transport connection. When the transport connection is established, a CONNECT SPDU is sent on the transport normal flow. The SPM then waits until it receives an ACCEPT, REFUSE or an ABORT SPDU.

4.1.2 Receiving the CONNECT SPDU

A valid incoming CONNECT SPDU results in an S-CONNECT.indication. The SS user is selected according to the destination address parameter of the CONNECT SPDU. The SPM then waits for an S_CONNECT response or an S_U_ABORT request from the called SS user, or an ABORT SPDU from the initiator.

4.2 ACCEPT SPDU

An SPM receiving a CONNECT SPDU may accept a proposal to establish a session connection by transferring an ACCEPT SPDU to the initiator, on the same transport connection.

The ACCEPT SPDU contains:

1. A Connection Identifier parameter group, provided to assist the SS users in identifying this specific session connection. This parameter has no effect on the SPM. The same considerations which are valid for the CONNECT SPDU apply here. This PGI has the same structure as that one specified in the corresponding CONNECT SPDU.

2. The session addresses associated with the initiator and acceptor are given back **as they have been coded in the CONNECT SPDU.**

3. The Connect/Accept parameter group, which contains:

A. A Protocol Version parameter, which is set to 1 (one).

B. A protocol Options parameter, which is set to 1 (one). This means that the concatenation is always supported.

C. A TPDU maximum size parameter, which is always set to 0 (zero), to indicate that segmenting is not supported.

D. In the case where synchronization is required on the session connection, the accepting SPM provides the initial value of the serial number which is to be used over the session connection.

E. Initial token settings will be supplied for each token available on this connection, as derived from the functional unit bits in the accepted Session requirements parameters.

4. Up to 512 octets of user data that can be passed from the accepting SS user to the initiating SS user.

5. The Session Requirements parameter, which contains a bitmap identifying the SS users supported functional units. This bitmap, contained in the ACCEPT SPDU, may indicate the support of functional units which were not proposed in the Session Requirements parameter of the CONNECT SPDU. In this case, only the functional units common to both the CONNECT and the ACCEPTS SPDUs are available over the established Session connection.

Each functional unit indicated as requested in the CONNECT SPDU is subject to negotiation between the two SS users. If the availability of a token controlled functional unit is agreed upon, then in the case where the initiator has indicated that the initial assignment of the related token is negotiable (acceptor's choice), the accepting SS user must fix this assignment.

4.2.1 Sending the ACCEPT SPDU

An S-CONNECT(accept).response results in an ACCEPT SPDU. This SPDU is sent on the transport normal flow. The SPM enters the data transfer phase and can receive any service request or SPDU that is allowed by the available functional units and current token positions. The SPM sets the variable V(M), which contains the serial number to be used for the first mark, to the value given by the called SS user. V(R) is set to zero; V(A) is set to V(M).

4.2.2 Receiving the ACCEPT SPDU

A valid incoming ACCEPT SPDU results in an S-CONNECT(accept).confirmation. The user data is passed to the SS user. The SPM sets the variable V(M), which contains the serial number to be used for the next minor or major synchronization point, to the value contained in the ACCEPT SPDU. V(R) is set to zero; V(A) is set to V(M). If the called SS user has requested any tokens in the Token item parameter of the ACCEPT SPDU, an S-PLEASE-TOKEN request is generated.

4.3 REFUSE SPDU

A CONNECT SPDU is rejected by transfer of either a REFUSE SPDU or an ABORT SPDU by the acceptor, instead of an ACCEPT SPDU.

4.3.1 Content of REFUSE SPDU

The REFUSE SPDU contains:

1. Connection Identifier parameter, which enables the SS users to identify this specific session connection. This parameter has no effect on the SPM. The same considerations which are valid for the CONNECT SPDU apply here. This PGI has the same structure as that one specified in the corresponding CONNECT SPDU.

2. Reason code parameter, giving the reason for refusal of the attempt to establish a session connection, together with a limited amount of transparent user data.

3. Transport Disconnect parameter, which indicates whether or not the connection is to be kept. In OSIRIDE version 1, this parameter **always requires the release of the Transport connection.**

4. Version Number parameter to identify which versions of this protocol have been implemented by the sending SPM. In OSIRIDE version 1, this value **is always 1 (one).**

5. Session User Requirements parameter, which contains a list of the functional units supported by the sending SPM, and required by the called SS user.

4.3.2 Sending the REFUSE SPDU

An S-CONNECT(reject).response results in a REFUSE SPDU. This SPDU is sent on the transport normal flow. The session connection ceases to exist. As the Transport Disconnect parameter always requires release of the Transport connection, the SPM goes to a state expecting a T_DISCONNNECT indication.

4.3.3 Receiving the REFUSE SPDU

A valid incoming REFUSE SPDU results in an S-CONNECT(reject).confirmation. The session connection ceases to exist. As the Transport Disconnect parameter always indicates that a release of the transport connection is requested, the SPM releases the transport connection, by making a T-DISCONNECT request.

4.4 FINISH SPDU

Orderly release is initiated by transfer of a FINISH SPDU, which may be transferred at any time during the data transfer phase. It requests as a response either:

1. a DISCONNECT SPDU to complete the release of the session connection or

2. a NOT FINISHED SPDU to refuse the release of the session connection if the release token is available.

The FINISH SPDU is transferred in sequence with any normal data being transferred. The right to issue a FINISH SPDU is restricted to the owner of all available tokens.

4.4.1 Content of FINISH SPDU

The FINISH SPDU contains:

1. Transport Disconnect parameter, which indicates whether or not the transport connection is to be kept. In OSIRIDE version 1, this parameter **always requires transport disconnection.**

2. User data parameter, which allows a limited amount of transparent user data to be transferred.

4.4.2 Sending the FINISH SPDU

An S-RELEASE.request results in a FINISH SPDU. This SPDU is sent on the transport normal flow.

After transferring a FINISH SPDU, the SPM should not transfer further SPDUs (except ABORT SPDU), unless a NOT FINISHED SPDU is received, after which the data transfer phase may be resumed. Receipt of a DISCONNECT SPDU signals completion of orderly session release.

4.4.3 Receiving the FINISH SPDU

A valid incoming FINISH SPDU results in an S-RELEASE.indication. The user data is passed to the SS user. The SPM waits for an S-RELEASE.response.

4.5 DISCONNECT SPDU

After receipt of a FINISH SPDU, a DISCONNECT SPDU may be transferred. Receipt of a DISCONNECT SPDU after transferring a FINISH SPDU signals the orderly release of the session connection. The DISCONNECT SPDU is transferred in sequence with any normal data being transferred.

4.5.1 Content of DISCONNECT SPDU

The DISCONECT SPDU contains a User data parameter, which allows a limited amount of data to be transferred.

4.5.2 Sending the DISCONNECT SPDU

An S-RELEASE(affirmative).response results in a DISCONNECT SPDU. This SPDU is sent on the transport normal flow. The session connection ceases to exist. The SPM then waits for a T_DISCONNECT indication.

4.5.3 Receiving the DISCONNECT SPDU

A valid incoming DISCONNECT SPDU results in an S-RELEASE(affirmative).confirmation. The session connection ceases to exist.

The transport connection is never re-used, that is, a TRANSPORT-DISCONNECT.request is issued.

4.6 ABORT SPDU

The ABORT SPDU is used to reject a session connection establishment attempt, or to cause abnormal release of a session connection at any time. This SPDU is also used by an SPM to release the session connection when a protocol error is detected. The ABORT SPDU **always** cause the release of the transport connection. Use of the ABORT SPDU may cause loss of data.

4.6.1 Content of the ABORT SPDU

The ABORT SPDU contains:

1. Transport Disconnect parameter, which indicates whether or not the transport connection is to be kept. In OSIRIDE version 1, this parameter **always requests that the transport connection is released.**

2. Reflect Parameter Value parameter, which, if present, allows implementation defined information to be trasnferred.

3. User Data parameter, which allows a limited amount of transparent user data to be transferred.

4.6.2 Sending the ABORT SPDU

An S-U-ABORT.request, or the detection of a protocol error in any state of the SPM results in an ABORT SPDU. The SPDU is sent on the transport expedited flow. The SPM starts the abort timer, TAB, and waits for:

1. An ABORT ACCEPT SPDU. This SPDU is accepted **only** for compatibility, and **shall never be used** in the first OSIRIDE version.

2. A transport disconnect indication, after which the TAB timer will be reset.

3. The expiration of the TAB timer, after which transport disconnection is requested.

4.6.3 Receiving the ABORT SPDU

A valid incoming ABORT SPDU results in an S-U-ABORT.indication or an S-P-ABORT.indication, depending on whether the abort is user generated or provider generated. The session connection ceases to exist. The transport connection shall be released by the SPM.

4.7 ABORT ACCEPT SPDU

The ABORT ACCEPT SPDU is not used in the first OSIRIDE implementation, but is supported on reception for compatibility.

4.7.1 Content of ABORT ACCEPT SPDU

The ABORT ACCEPT SPDU contains no parameters.

4.7.2 Sending the ABORT ACCEPT SPDU

The ABORT ACCEPT SPDU will never be sent by an OSIRIDE implementation (version 1).

4.7.3 Receiving the ABORT ACCEPT SPDU

A valid incoming ABORT ACCEPT SPDU results in resetting the abort timer, TAB, and releasing the transport connection. The session connection ceases to exist.

4.8 NOT FINISHED SPDU

After receipt of a FINISH SPDU, a NOT FINISHED SPDU may be transferred only if the release token is available. No confirmation is sought.

4.8.1 Content of NOT FINISHED SPDU

The NOT FINISHED SPDU contains a User Data parameter, which allows a limited amount of transparent user data to be transferred.

4.8.2 Sending the NOT FINISHED SPDU

An S-RELEASE(negative).response results in a NOT FINISHED SPDU. This SPDU is sent on the transport normal flow. The SPM remains in the data transfer phase and can receive any service request or SPDU that is allowed by the available functional units and current token positions.

4.8.3 Receiving the NOT FINISHED SPDU

A valid incoming NOT FINISHED SPDU results in an S-RELEASE(negative).confirmation. The SPM remains in the data transfer phase and can receive any service request or SPDU that is allowed by the available functional units and current token positions.

4.9 DATA TRANSFER SPDU

Normal data is transferred by use of the DATA TRANSFER SPDU. As the concatenation option **is always selected** during connection establishment, certain concatenations of the DATA TRANSFER SPDU with other SPDUs is allowed (see "3.4.4 Concatenation" pag. 25).

If the DATA TOKEN is available for the session connection, only the owner of this token may issue DATA TRANSFER SPDUs.

4.9.1 Content of DATA TRANSFER SPDU

The DATA TRANSFER SPDU contains:

1. Enclosure Item parameter, to indicate the end of SSDU when segmenting has been selected. This parameter is not used in OSIRIDE Version 1.

2. User Information Field, to transfer an unlimited amount of transparent user data.

4.9.2 Sending the DATA TRANSFER SPDU

An S-DATA.request results always in a DATA TRANSFER SPDU, because segmenting is not selected. DATA TRANSFER SPDUs are sent on the transport normal flow.

4.9.3 Receiving the DATA TRANSFER SPDU

A valid incoming DATA TRANSFER SPDU results in an S-DATA.indication, because segmenting is not selected.

4.10 GIVE TOKENS SPDU

The GIVE TOKENS SPDU is used:

1. to cause assignement of tokens to be changed and/or

2. to introduce a concatenated sequence of SPDUs.

If the GIVE TOKENS SPDU does not contain a parameter field, it is used to indicate concatenation without assignement of tokens and, in this case, the following procedures do not apply.

4.10.1 Content of GIVE TOKENS SPDU

The GIVE TOKENS SPDU contains a Token Item parameter, which indicates which tokens are being transferred from the sending SS user to the receiving SS user.

4.10.2 Sending the GIVE TOKENS SPDU

An S-TOKEN-GIVE.request results in a GIVE TOKENS SPDU. This SPDU is sent on the transport normal flow.

4.10.3 Receiving the GIVE TOKENS SPDU

A valid incoming GIVE TOKENS SPDU results in a S-TOKEN-GIVE.indication.

4.11 PLEASE TOKENS SPDU

The PLEASE TOKENS SPDU is used:

1. to request that the token assignments be changed to permit the requestor to be authorized to perform a function associated with the requested tokens and/or

2. to introduce a concatenated sequence of SPDUs.

If the PLEASE TOKENS SPDU does not contain a parameter field, it is used to indicate concatenation without requesting tokens and, in this case, the following procedures do not apply.

4.11.1 Content of PLEASE TOKENS SPDU

The PLEASE TOKENS SPDU contains:

1. Token Item parameter, which indicates which tokens are being requested by the sending SS user.

2. User Data parameter, which allows a limited amount of transparent user data to be transferred.

4.11.2 Sending the PLEASE TOKENS SPDU

An S-TOKEN-PLEASE.request results in a PLEASE TOKENS SPDU. This SPDU is sent on the transport normal flow.

4.11.3 Receiving the PLEASE TOKENS SPDU

A valid incoming PLEASE TOKENS SPDU results in an S-TOKEN-PLEASE.indication.

Receiving a PLEASE TOKENS SPDU for tokens which are not currently assigned to the receiving SS user is not a protocol error.

4.12 TYPED DATA SPDU

The TYPED DATA SPDU enables the SS-users tc transmit transparent user data, irrespective of the availability or assignement of the data token. In all other respects, the same constraints apply as for normal data (see "4.9 DATA TRANSFER SPDU" pag. 38).

4.12.1 Content of TYPED DATA SPDU

The TYPED DATA SPDU contains:

1. Enclosure Item parameter, to indicate the beginning and end of SSDU when segmenting has been selected. In OSIRIDE Version 1, this parameter is not used.

2. User Information Field, to transfer transparent user data of unlimited size.

4.12.2 Sending the TYPED DATA SPDU

An S_TYPED_DATA request results in the transfer of a TYPDE DATA SPDU. Each SPDU is mapped in onto one TSDU. TYPED DATA SPDUs are sent on the transport normal flow. The current state of the SPM is not changed.

4.12.3 Receiving the TYPED DATA SPDU

A valid incomind TYPED DATA SPDU results in an S_TYPED_DATA indication. The current state of the SPM is not changed.

4.13 MINOR SYNCHRONIZATION POINT SPDU

The MINOR SYNCHRONIZATION POINT SPDU is used to send a minor synchronization point. A confirmation may be returned by the receiver, but is not required by the SPM. All acknowledgements rules are defined by the SS users. In particular, whether confirmation is requested or not is transparent to the SPM. Only the owner of the synchronize-minor token and the data token (if available) may issue a MINOR SYNCHRONIZATION POINT SPDU.

4.13.1 Content of MINOR SYNCHRONIZATION POINT SPDU.

The MINOR SYNCHRONIZATION POINT SPDU contains:

1. Sync Type Item parameter, which is used to indicate if an explicit confirmation is required.

2. Serial Number parameter, which indicates the serial number of this minor synchronization point, and is set by the SPM to the current value of **V(M)**.

3. User Data parameter, which allows a limited amount of transparent user data to be transferred.

4.13.2 Sending the MINOR SYNCHRONIZATION POINT SPDU

An S-SYNC-MINOR.request results in a MINOR SYNCHRONIZATION POINT SPDU. This SPDU is sent on the transport normal flow. If **V-SC** is true, **V(A)** is set equal to **V(M)**, and **V-SC** is set false. **V(M)** is incremented by 1.

4.13.3 Receiving the MINOR SYNCHRONIZATION POINT SPDU

A valid incoming MINOR SYNCHRONIZATION POINT SPDU (when the received serial number equals **V(M)**) results in an S-SYNC-MINOR.indication. If **V-SC** is false, **V(A)** is set equal to **V(M)**, and **V-SC** is set true. **V(M)** is incremented by 1.

4.14 MINOR SYNC ACK SPDU

The MINOR SYNC ACK SPDU is used to return a confirmation of MINOR synchronization points. The SPM sends a MINOR SYNC ACK SPDU for each S-SYNC-MINOR.response.

4.14.1 Content of MINOR SYNC ACK SPDU

The MINOR SYNC ACK SPDU contains:

1. Serial Number parameter, provided by the SS user, which indicates the serial number of the minor synchronization point point which is being confirmed.

2. User Data parameter, which allows a limited amount of transparent user data to be transferred.

4.14.2 Sending the MINOR SYNC ACK SPDU

An S-SYNC-MINOR.response (with **V-SC** true and received serial number greater than or equal to **V(A)** and less than **V(M)**) results in sending a MINOR SYNC ACK SPDU. This SPDU is sent on the transport normal flow. The SPM sets **V(A)** equal to the serial number plus one.

4.14.3 Receiving the MINOR SYNC ACK SPDU

A valid incoming MINOR SYNC ACK SPDU (with **V-SC** true and received serial number greater than or equal to **V(A)** and less than **V(M)**) results in an S-SYNC-MINOR.confirmation. The SPM sets **V(A)** equal to the received serial number plus one.

4.15 MAJOR SYNCHRONIZATION POINT SPDU

The MAJOR SYNCHRONIZATION POINT SPDU is used to send a MAJOR mark at a major synchronization point. A confirmation has to be received before more data can be sent on the normal and expedited flows. Only the owner of the major token (and data token and synchronize minor token, if available) has the right to issue a MAJOR SYNCHRONIZATION POINT SPDU.

4.15.1 Content of MAJOR SYNCHRONIZATION POINT SPDU

The MAJOR SYNCHRONIZATION POINT SPDU contains:

1. Sync Type Item parameter, which is only present when indicating that this major synchronization point is not the end of the current activity. In OSIRIDE version 1, **this parameter is always present.**

2. Serial Number parameter, which indicates the serial number of this major synchronization point, and is set by the SPM to the current value of **V(M)**.

3. User Data parameter, which allows a limited amount of transparent user data to be transferred.

4.15.2 Sending the MAJOR SYNCHRONIZATION POINT SPDU

An S-SYNC-MAJOR.request results in a MAJOR SYNCHRONIZATION POINT SPDU. This SPDU is sent on the transport normal flow.

If **V-SC** is true, **V(A)** is set equal to **V(M)**, and **V-SC** is set to false. **V(M)** is incremented by 1. The SPM waits for a PREPARE (MAJOR SYNC ACK) SPDU, on the transport expedited flow, followed by a MAJOR SYNC ACK SPDU on the transport normal flow.

Any other SPDUs received prior to the MAJOR SYNC ACK SPDU will result in the appropriate service indications being given to the SS user.

4.15.3 Receiving the MAJOR SYNCHRONIZATION POINT SPDU

A valid incoming MAJOR SYNCHRONIZATION POINT SPDU (with received serial number equal to V(M)) results in an S-SYNC-MAJOR.indication. If **V-SC** is false, **V(A)** is set equal to **V(M)**. **V(M)** is incremented by 1.

4.16 MAJOR SYNC ACK SPDU

The MAJOR SYNC ACK SPDU is used to return a confirmation to MAJOR synchronization points.

4.16.1 Content of the MAJOR SYNC ACK SPDU

The MAJOR SYNC ACK SPDU contains:

1. Serial Number parameter, which indicates the serial number of the major synchronization point which is being confirmed, and is equal to V(M) minus 1.

2. User Data parameter, which allows a limited amount of user data to be transferred.

4.16.2 Sending the MAJOR SYNC ACK SPDU

An S-SYNC-MAJOR.response results in sending a MAJOR SYNC ACK SPDU. This SPDU is sent on the transport normal flow. Before that, a PREPARE SPDU is sent on the transport expedited flow. V(A) and V(R) are set to V(M).

4.16.3 Receiving the MAJOR SYNC ACK SPDU

A valid incoming MAJOR SYNC ACK SPDU (with V-SC false and received serial number equal to V(M) minus one) results in an S-SYNC-MAJOR.confirmation.

Two successive SPDUs will be received:

- PREPARE (MAJOR SYNC ACK) SPDU on the transport expedited flow, followed by

- MAJOR SYNC ACK SPDU on the transport normal flow.

V(A) and V(R) are set to V(M).

4.17 RESYNCHRONIZE SPDU

The RESYNCHRONIZE SPDU is used to provide the SS users with a selective means to resynchronize the exchange of data to a previous synchronization point and to reposition the tokens to an agreed side. This procedure may result in loss of data.

This SPDU can also be used to "purge" the session connection, since that is a particular case of resynchronization. The following options are provided:

1. Abandon

2. Set

3. Restart

Since the resynchronization protocol provides a repositioning of the tokens, a particular use of it is the destructive way to get the tokens.

4.17.1 Content of RESYNCHRONIZE SPDU

The RESYNCHRONIZE SPDU contains:

1. Token Setting Item parameter, which indicates the requestor's proposed token positions for all available tokens.

2. Resync Type Item parameter, which indicates the resynchronize option (abandon, set or restart).

3. Serial Number parameter, which indicates the serial number to which resynchronization is being requested. The serial number is supplied by the SS user if the resynchronize option is restart or set. If the resynchronize option is abandon, the serial number is set to the value of V(M) of the sending SPM.

4. User Data parameter, which allows a limited amount of transparent user data to be transferred.

4.17.2 Sending the RESYNCHRONIZE SPDU

An S-RESYNCHRONIZE.request (with serial number greater than or equal to V(R) and less than V(M)) results in a RESYNCHRONIZE SPDU. This SPDU is sent on the transport normal flow. A PREPARE SPDU is sent before that on the transport expedited flow. The SPM goes into a state where all the incoming SPDUs are discarded, except PREPARE (RESYNCHRONIZE), PREPARE (RESYNCHRONIZE ACK), RESYNCHRONIZE and ABORT. The receipt of these SPDUs initiates a resynchronization contention, which is dealt with in "4.17.4 Resynchronization contention."

4.17.3 Receiving the RESYNCHRONIZE SPDU

Except when a resynchronization contention has occurred, a valid incoming RESYNCHRONIZE SPDU (if the resynchronize option is restart, with serial number greater than or equal to V(R) and less then V(M)) results in an S-RESYNCHRONIZE.indication.

Two successive SPDUs will be received:

- PREPARE (RESYNCHRONIZE) SPDU on the transport expedited flow, followed by
- RESYNCHRONIZE SPDU on the transport normal flow.

When the PREPARE (RESYNCHRONIZE) SPDU is received, all subsequently received SPDUs, except ABORT SPDU, are discarded until the RESYNCHRONIZE SPDU is received on the transport normal flow. The SPM now waits for an S-RESYNCHRONIZE.response. If a resynchronization contention has occurred, only the contention loser (see "4.17.4 Resynchronization contention" pag. 47) passes an S-RESYNCHRONIZE indication to the SS-user.

4.17.4 Resynchronization contention

The contention between two RESYNCHRONIZE SPDUs is resolved according to the table in Fig. 20 pag. 47. The table defines the contention winner whose SPDU is taken into account; the other SPDU is discarded.

If the incoming RESYNCHRONIZE SPDU is not acceptable, the receiving SS user may issue another if it prevails over the original proposal according to the decision rules.

Outgoing SPDU from SPM A	Incoming SPDU from SPM B		
	RESYNC. (abandon)	RESYNC. (set)	RESYNC. (restart)
RESYNC. (abandon)	initiator	SPM A	SPM A
RESYNC. (set)	SPM B	initiator	SPM A
RESYNC. (restart)	SPM B	SPM B	SPM with lowest serial number or if equal then initiator

Fig. 20. Contention winner

4.18 RESYNCHRONIZE ACK SPDU

The RESYNCHRONIZE ACK SPDU is used to notify the sender of a RESYNCHRONIZE SPDU of the completion of resynchronization.

4.18.1 Content of RESYNCHRONIZE ACK SPDU

The RESYNCHRONIZE ACK SPDU contains:

1. Token Setting parameter, which indicates the selected token positions.

2. Serial Number parameter, which indicates the first serial number to be used in the resynchronized flow. This parameter is set according to the Resync Type Item parameter in the RESYNCHRONIZE SPDU:

   A. For the **restart** option, to the serial number in the RESYNCHRONIZE SPDU.

   B. For the **set**, to the serial number in the S-RESYNCHRONIZE response.

   C. For the **abandon**, to the serial number in the RESYNCHRONIZE SPDUs, or to V(M), whichever is higher.

3. User Data parameter, which allows a limited amount of transparent user data to be transferred.

4.18.2 Sending the RESYNCHRONIZE ACK SPDU

An S-RESYNCHRONIZE.response results in sending a RESYNCHRONIZE ACK SPDU. This SPDU is sent on the transport normal flow. Before that, a PREPARE SPDU is sent on the transport expedited flow.

The tokens are set to the values proposed by the requestor. If the requestor has indicated "accepting SS user's choice" for a token, then the acceptor's proposed value for that token is used. The selected token settings are returned in the Token Setting Item of the RESYNCHRONIZE ACK SPDU.

V(A) and **V(M)** are set to the serial number contained in the RESYNC ACK SPDU.

V(R) is unchanged if the Resync Type Item parameter in the received RESYNCHRONIZE SPDU indicated the **restart** option. Otherwise V(R) is set to zero.

4.18.3 Receiving the RESYNCHRONIZE ACK SPDU

A valid incoming RESYNCHRONIZE ACK SPDU results in an S-RESYNCHRONIZE.confirmation. Two successive SPDUs will be received:

A. PREPARE (RESYNCHRONIZE ACK) SPDU on the transport expedited flow, followed by

B. RESYNCHRONIZE ACK on the transport normal flow.

The tokens are set to the position indicated in the RESYNCHRONIZE ACK SPDU.

V(A) and **V(M)** ar set to the serial number contained in the RESYNCHRONIZE ACK SPDU.

V(R) is unchanged if the Resync Type Item parameter indicated the **restart** option. Otherwise, **V(R)** is set to 0.

4.19 PREPARE SPDU

The PREPARE SPDU is used to notify the imminent arrival of certain SPDUs and indicates to the receiving SPM that SPDUs received on the transport normal flow may be discarded under certain circumstances.

4.19.1 Content of PREPARE SPDU

The PREPARE SPDU contains a Prepare Type parameter which indicates which SPDU should be expected on the transport normal flow.

4.19.2 Sending the PREPARE SPDU

The PREPARE SPDU is sent before the associated SPDUs specified in Fig. 21 pag. 50. The same table specifies the value of the Prepare Type parameter.

Associated SPDU	Prepare Type
RESYNCHRONIZE SPDU	RESYNCHRONIZE
RESYNCHRONIZE ACK SPDU	RESYNCHRONIZE ACK
MAJOR SYNC ACK SPDU	MAJOR SYNC CONFIRMATION

Fig. 21. SPDUs associated with the PREPARE SPDU

The PREPARE SPDU is sent on the transport expedited flow (its associated SPDU being sent on the transport normal flow). The SPM goes to a state which is determined by the initial request.

4.19.3 Receiving the PREPARE SPDU

A valid incoming PREPARE SPDU results in the SPM entering a state where it is waiting for the associated SPDU on the transport normal flow. If the Prepare Type parameter indicates MAJOR SYNC ACK, any SPDUs received on the transport normal flow are actioned normally. Otherwise, SPDUs received on the transport normal flow before the indicated SPDU will be discarded. In any case, an EXPEDITED DATA SPDU which is received after a PREPARE SPDU, but before the associated SPDU on the transport normal flow, is not passed to the SS user until the associated SPDU has been received and actioned.

5.0 STRUCTURE AND ENCODING OF SPDUS

5.1 TSDU STRUCTURE

Each TSDU consists of one or more SPDUs complying with the requirements for concatenation (see "3.4.4 Concatenation" pag. 25). Each SPDU within a TSDU consists of one or more octets that are numbered sequentially starting from 1.

Bits in each octet shall be numbered 8 to 1, where 1 is the low order bit. Where two octets are used, the bits shall be numbered from 16 to 1, where 1 is the low order bit.

5.2 SPDU STRUCTURE

5.2.1 SPDUs

SPDUs shall contain, in the following order:

A. the SI field that identifies the type of SPDU;

B. the LI field that indicates the length of the associated parameter field;

C. the parameter field which, if present, consists of the PGI units and/or the PI units defined for the SPDU;

D. the user information field, if defined for the SPDU and if present.

5.2.2 PGI units

PGI units shall contain, in the following order:

A. the PGI field, that identifies the parameter group;

B. the LI field, that indicates the length of the associated parameter field;

C. the parameter field, which, if present, consists of either

a. a single parameter value [3]

b. one or more PI units (see "5.2.3 PI units" pag. 52).

5.2.3 PI units

PI units shall contain, in the following order:

A. the PI field, that identifies the parameter;

B. the LI field, that indicates the length of the associated parameter field;

C. the parameter field, which, if present, consists of the parameter value.

5.2.4 Length indicator field

The value of the LI field is expressed as a binary number representing the length, in octets, of the associated parameter field[4]. A value of zero indicates that the associated parameter field is absent.

LI fields indicating lengths within the range 0-254 shall comprise one octet.

LI fields indicating lengths within the range 255-65535 shall comprise three octets. The first octet shall be coded 1111 1111 and the second and third octets shall contain the length of the associated parameter field with the high order bits in the first of these two octets.

5.2.5 Parameter fields

PI units and PGI units defined as mandatory in the tables in "5.3 SPDU identifiers and associated parameter fields" pag. 55 shall contain parameter fields of one or more octets.

Any PGI units or PI units defined as non mandatory may be omitted if it is not required for conveying information (i.e. parameter value).

---

[3] A PGI with one parameter is structurally equivalent to a PI unit.

[4] The value of the LI field does not include either itself or any subsequent user information.

If a PGI unit or PI unit contains an LI field with the value zero, the associated parameter field is absent. The value of the parameter field shall be considered as its defaul value.

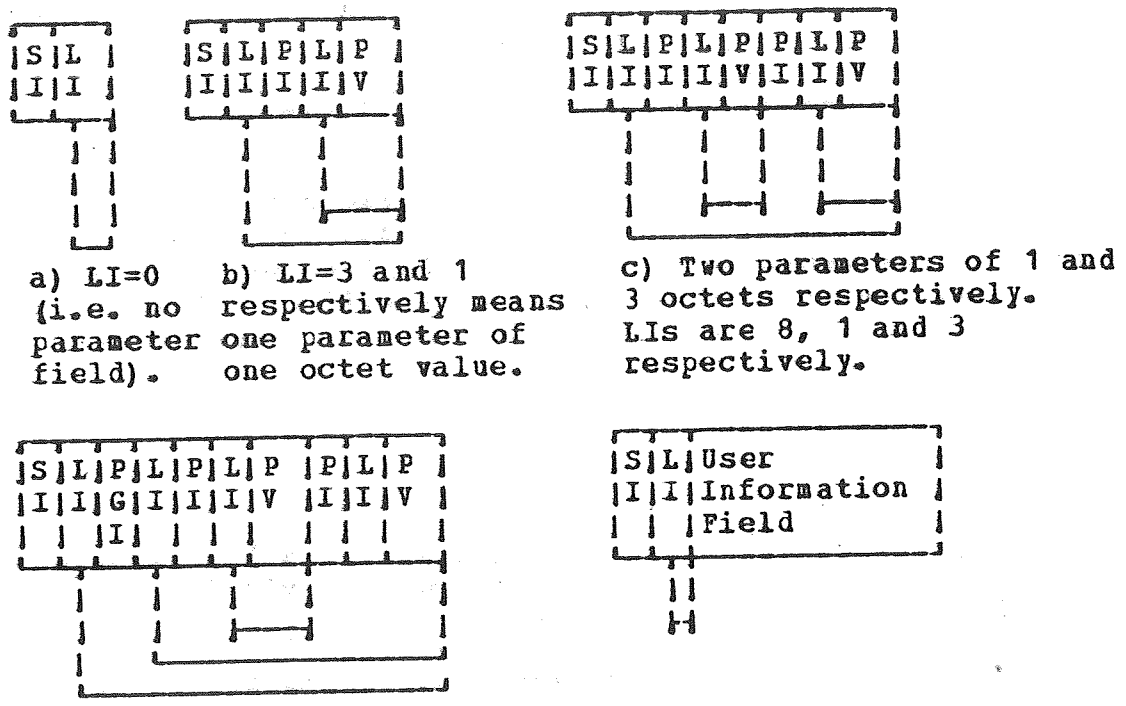
Note: In OSIRIDE, if a non mandatory parameter is absent, the associated PGI (or PI) and LI fields shall not be included in the SPDU. PGI units and PI units within the same nesting level shall be ordered in increasing value of their PGI and PI codes.

In the tables of "5.3 SPDU identifiers and associated parameter fields" pag. 55, PGI codes and PI codes are expressed as decimal numbers.

5.2.6 Parameter values

Parameter values which are bit encoded, with bits which are indicated as reserved shall have those bits set to zero in the SPDU which is sent. The reserved bits may be ignored in a received SPDU.

Fig. 22 pag. 53 gives an example of the encoding structure.



a) LI=0 (i.e. no parameter field).

b) LI=3 and 1 respectively means one parameter of one octet value.

c) Two parameters of 1 and 3 octets respectively. LIs are 8, 1 and 3 respectively.

d) One PGI with two enclosed PIs.

e) The most simple carrier of information.

Fig. 22. Examples of SPDU structure

b. one or more PI units (see "5.2.3 PI units" pag. 52).

5.2.3 PI units

PI units shall contain, in the following order:

A. the PI field, that identifies the parameter;

B. the LI field, that indicates the length of the associated parameter field;

C. the parameter field, which, if present, consists of the parameter value.

5.2.4 Length indicator field

The value of the LI field is expressed as a binary number representing the length, in octets, of the associated parameter field[4]. A value of zero indicates that the associated parameter field is absent.

LI fields indicating lengths within the range 0-254 shall comprise one octet.

LI fields indicating lengths within the range 255-65535 shall comprise three octets. The first octet shall be coded 1111 1111 and the second and third octets shall contain the length of the associated parameter field with the high order bits in the first of these two octets.

5.2.5 Parameter fields

PI units and PGI units defined as mandatory in the tables in "5.3 SPDU identifiers and associated parameter fields" pag. 55 shall contain parameter fields of one or more octets.

Any PGI units or PI units defined as non mandatory may be omitted if it is not required for conveying information (i.e. parameter value).

---

[3] A PGI with one parameter is structurally equivalent to a PI unit.

[4] The value of the LI field does not include either itself or any subsequent user information.

If a PGI unit or PI unit contains an LI field with the value zero, the associated parameter field is absent. The value of the parameter field shall be considered as its defaul value.

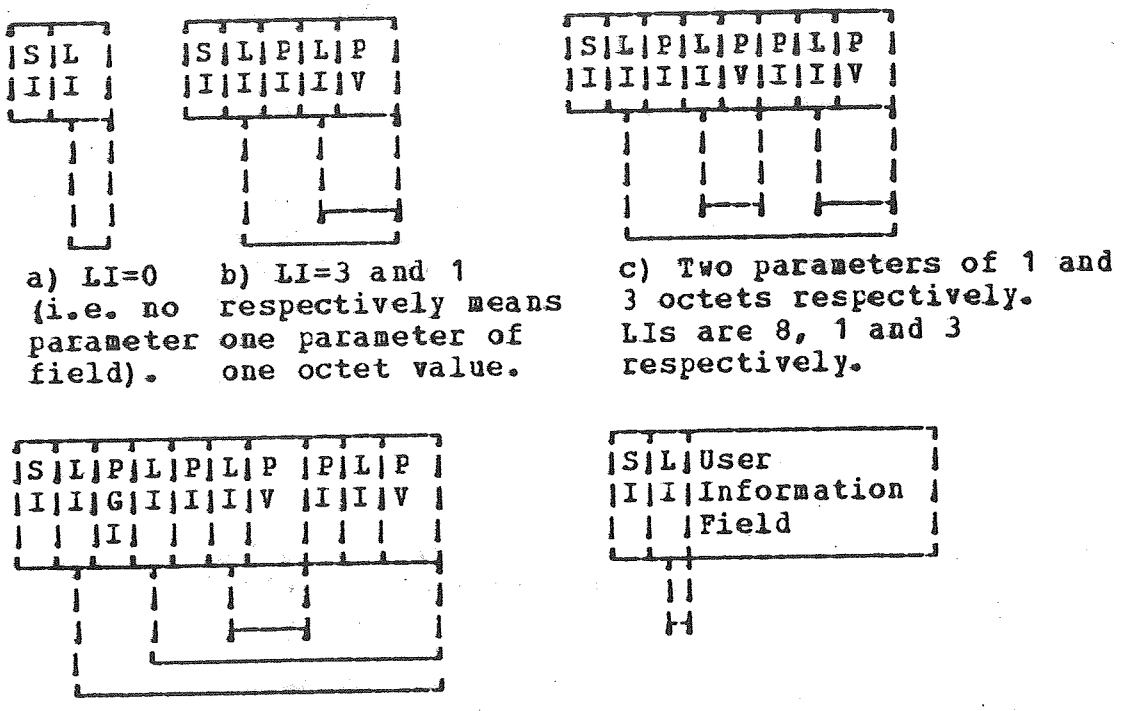
Note: In OSIRIDE, if a non mandatory parameter is absent, the associated PGI (or PI) and LI fields shall not be included in the SPDU. PGI units and PI units within the same nesting level shall be ordered in increasing value of their PGI and PI codes.

In the tables of "5.3 SPDU identifiers and associated parameter fields" pag. 55, PGI codes and PI codes are expressed as decimal numbers.

5.2.6 Parameter values

Parameter values which are bit encoded, with bits which are indicated as reserved shall have those bits set to zero in the SPDU which is sent. The reserved bits may be ignored in a received SPDU.

Fig. 22 pag. 53 gives an example of the encoding structure.



a) LI=0 (i.e. no parameter field).

b) LI=3 and 1 respectively means one parameter of one octet value.

c) Two parameters of 1 and 3 octets respectively. LIs are 8, 1 and 3 respectively.

d) One PGI with two enclosed PIs.

e) The most simple carrier of information.

Fig. 22. Examples of SPDU structure

5.2.7 User Information field

Successive sequences of segmented SSDUs are contained in the user information field of successive SPDUs. Non-segmented SSDUs are contained in the user information field of a single SPDU. The order of the octets and the order of the bits in the SSDU is maintained in the SPDUs.

5.3 SPDU IDENTIFIERS AND ASSOCIATED PARAMETER FIELDS

In the following sections, the encoding of the OSIRIDE SPDUs is described. For each SPDU, some parameters are mandatory (■), others are not mandatory (■■). The text describing the protocol procedures indicates which non mandatory parameters are used in OSIRIDE Version 1.

5.3.1 CONNECT (CN) SPDU

PGI	m/nm	Code	PI	m/nm	Code	Length/Value
Connection Identifier	m	1	Calling SS-user reference	nm	10	24 octets maximum
			Common Reference	nm	11	14 octets maximum
			Additional Reference Information	nm	12	2 octets maximum
Connect/ Accept Item	nm	5	Protocol Options	m	19	1 octet
			TSDU max size	nm	21	4 octets
			Version Number	m	22	1 octet
			Initial Serial Number	nm	23	6 octets maximum
			Token Setting Item	nm	26	1 octet
			Session User Requirements	nm	20	2 octets
			Calling SSAP Identifier	nm	51	16 octets maximum
			Called SSAP Identifier	nm	52	16 octets maximum
User Data	nm	193				512 octets maximum

Fig. 23. Encoding of CONNECT SPDU

Notes:

1. The SI field shall contain the value 13.

2. The Connection Identifier PV field shall be as defined by the calling SS user.

3. If the Connect/Accept Item or any of the associated PI are absent, the following default values shall apply:

   A. Protocol Options: SPDU with extended concatenation cannot be received.

   B. TSDU Maximum size: segmenting is not proposed.

   C. Version Number: this version is proposed

   D. Initial Serial NUmber: shall have the value 0.

   E. Token Item: all tokens whose availability is proposed in the Session User Requirements parameter are assigned to the calling SS user.

4. The Protocol Options PV field shall indicate whether or not the initiator is able to receive extended concatenated SPDUs. The coding for that field shall be:

   A. bit 1 = 1 able to receive extended concatenated SPDUs

   B. bit 1 = 0 not able to receive extended concatenated SPDUs

   Bits 2-8 are reserved (set to 0).

5. The TSDU Maximum Size parameter shall be present if the user of segmenting is proposed. In OSIRIDE **it shall not be present.**

6. In the Version Number PV field bit 1 shall have the value 1, indicating that this version is implemented.

7. Each digit of the serial number is encoded as an octet, as follows:

   A. 0 : 0

   B. 1 : 0

   C. 2 : 0

   D. 3 : 0

   E. 4 : 0

   F. 5 : 0

   G. 6 : 0

   H. 7 : 0

   I. 8 : 0

   J. 9 : 0

Serial number can range from 0 to 999999. The most significant digit is encoded first in the PV field. Leading zeros may be omitted.

8. The bits of the Token Setting Item PV field are defined as bit pairs:

   A. bits 8,7 release token

   B. bits 6,5 major token

   C. bits 4,3 synchronize-minor token

   D. bits 2,1 data token

   The coding for each bit pair shall be:

   A. 00 initiator's side;

   B. 01 responder's side;

   C. 10 called SS user's choice;

   D. 11 reserved.

   The values are relevant only if the appropriate Functional Units are requested in ther Session User Requirements parameter. If no Functional Unit requiring a token has been requested, this parameter need not be present.

9. The bits in the Session User Requirements PV field shall indicate the Functional Units proposed by the calling SS user, for use over this session connection:

   A. bit 1 ==> half duplex functional unit

   B. bit 2 ==> duplex functional unit

   C. bit 3 ==> expedited data functional unit **(never set)**

   D. bit 4 ==> minor synchronize functional unit

   E. bit 5 ==> major synchronize functional unit

   F. bit 6 ==> resynchronization functional unit

   G. bit 7 ==> activity management functicnal unit **(never set)**

   H. bit 8 ==> negotiated release functional unit

   I. bit 9 ==> capability data functional unit **(never set)**

   J. bit10 ==> exceptions functional unit **(never set)**

   K. bit11 ==> typed data functional unit

bits 12-16 reserved (set to 0).

The coding for each bit shall be:

0 ==> use of Functional Unit not proposed

1 ==> use of Functional Unit proposed

When this parameter is absent, the default shall be as though bits 1, 4, 7, 9 and 10 are set to one, and the remaining bits are set to zero.

10. The Calling SSAP Identifier, if present, shall be defined by the calling SS user.

11. The Called SSAP Identifier, if present, shall be as defined by the calling SS user.

12. The User Data PV field shall contain any user data supplied by the calling SS user.

5.3.2 ACCEPT (AC) SPDU

PGI	m/nm	Code	PI	m/nm	Code	Length/Value
Connection Identifier	m	1	Called SS-user reference	nm	9	24 octets maximum
			Common Reference	nm	11	14 octets maximum
			Additional Reference Information	nm	12	2 octets maximum
Connect/ Accept Item	nm	5	Protocol Options	m	19	1 octet
			TSDU max size	nm	21	4 octets
			Version Number	m	22	1 octet
			Initial Serial Number	nm	23	6 octets maximum
			Token Setting Item	nm	26	1 octet
			Token Item	nm	16	1 octet
			Session User Requirements	nm	20	2 octets
			Calling SSAP Identifier	nm	51	16 octets maximum
			Called SSAP Identifier	nm	52	16 octets maximum
User Data	nm	193				512 octets maximum

Fig. 24. Encoding of ACCEPT SPDU

Notes:

1. The SI field shall contain the value 14.

2. The Connection Identifier PV field shall be as defined by the called SS user.

3. If the Connect/Accept Item or any of the associated PI are absent, the following default values shall apply:

   A. Protocol Options: SPDU with extended concatenation cannot be received.

   B. TSDU Maximum size: segmenting is not selected.

   C. Version Number: this version is proposed

   D. Initial Serial NUmber: shall have the value 0.

   E. Token Item: all tokens whose availability is proposed in the Session User Requirements parameter are assigned to the calling SS user.

4. The Protocol Options PV field shall indicate whether or not the initiator is able to receive extended concatenated SPDUs. The coding for that field is defined in the CONNECT.

5. The TSDU Maximum Size parameter shall be present if the user of segmenting is proposed. In OSIRIDE **it shall not be present.**

6. The Version Number PV field encoding is dealt with in the CONNECT.

7. The encoding of the serial number is defined in the CONNECT.

8. The Token Setting Item PV field indicates the initial token settings for each token available on this session connection. The bits and encoding are defined in the CONNECT. In the case where the calling SS user has indicated that the initial assignement of the related token is negotiable (called SS user's choice), the called SS user must fix this assignement. Otherwise, the values set in the CONNECT SPDU must be returned. The value "called SS user's choice" is not a permitted value in the ACCEPT SPDU. The values are relevant only if the appropriate Functional Units are requested in the Session User Requirements parameter. If no Functional Unit requiring a token has been requested, this parameter need not be present.

9. The Token Item PV field shall indicate which tokens are requested by the called SS user:

   A. bit 7 = 1 release token

   B. bit 5 = 1 major token

   C. bit 3 = 1 synchronize minor token

   D. bit 1 = 1 data token

   Bits 2, 4, 6 and 8 are reserved.

   Bits corresponding to tokens which are not available are ignored.

10. The bits in the Session User Requirements PV field shall indicate the Functional Units proposed by the called SS user, for use over this session connection. This PV field shall not have both bit 1 set (half-duplex Functional Unit) and bit 2 set (duplex Functional Unit). The coding is defined as in the CONNECT SPDU.

11. The Calling SSAP Identifier, if present, shall be as defined be the calling SS user.

12. The Called SSAP Identifier, if present, shall be as defined by the called SS user.

13. The User Data PV field shall contain any user data supplied by the called SS user.

5.3.3 REFUSE (RF) SPDU

PGI	m/nm	Code	PI	m/nm	Code	Length/Value
Connection Identifier	m	1	Called SS-user reference	nm	9	24 octets maximum
			Common Reference	nm	11	14 octets maximum
			Additional Reference Information	nm	12	2 octets maximum
			Transport disconnect	nm	17	1 octet
			Session User Requirements	nm	20	2 octets
			Version Number	nm	22	1 octet
			Reas. code	nm	50	513 octets maximum

Fig. 25. Encoding of REFUSE SPDU

Notes:

1. The SI field shall contain the value 12.

2. The Connection Identifier PV field shall be as defined by the called SS user.

3. The Transport Disconnect PV field shall indicate whether or not the transport connection is to be kept. The coding for this field shall be:

   A. bit 1 = 0 transport connection is kept.

   B. bit 1 = 1 transport connection is released.

   Bits 2-8 are reserved.

   If this parameter is absent, the transport connection is released. **This parameter is absent in any OSIRIDE implementation.**

4. The Session User Requirements PV field shall indicate the Functional Units required by the called SS user and supported by the responder. The coding shall be the same as in the CONNECT SPDU.

5. The Version Number PV field shall have the same value and encoding as specified in the CONNECT SPDU.

6. The Reason Code PV field shall contain a reason code in the first octet. Depending on the value of this first octet, additional octets may be used. The following values are defined for the first octet:

   A. 0 ==> reason not specified

   B. 1 ==> rejection by called SS user due to a temporary congestion

   C. 2 ==> rejection by called SS user. The following octets may be used for up to 512 octets of user data

   D. * 128 + 1 ==> address SSAP unknown

   E. * 128 + 2 ==> SS user not attached to SSAP

   F. 128 + 3 ==> SPM congestion at connect time

   G. * 128 + 4 ==> proposed protocol version not supported

   Reasons marked with an asterisk (*) may be reported to the SS user as persistent, others reported as transient.

   All other values are reserved.

   The Session User Requirements parameter may only be present if the value of the reason code is 2. If the reason code has the value 2 and the Session User Requirements parameter is not

present, the default value shall be assumed (see "5.3.1 CONNECT (CN) SPDU" pag. 56).

5.3.4 FINISH (FN) SPDU

PGI	m/nm	Code	PI	m/nm	Code	Length/Value
			Transport disconnect	nm	17	1 octet
User data	nm	193				512 octets maximum

Fig. 26. Encoding of FINISH SPDU

Notes:

1. The SI field shall have the value 9.

2. The Transport Disconnect PV field shall indicate whether or not the transport connection is to be kept. The coding for this field shall be:

   A. bit 1 = 0 transport connection is kept.

   B. bit 1 = 1 transport connection is released.

   Bits 2-8 are reserved.

   If this parameter is absent, the transport connection is released. **This parameter is absent in any OSIRIDE implementation.**

3. The User Data PV field shall contain any user data supplied by the SS user.

5.3.5 DISCONNECT (DN) SPDU

PGI	m/nm	Code	PI	m/nm	Code	Length/Value
User data	nm	193				512 octets maximum

Fig. 27. Encoding of DISCONNECT SPDU

Notes:

1. The SI field shall have the value 10.
2. The User Data field shall contain any user data supplied by the SS user.

5.3.6 NOT FINISHED (NF) SPDU

PGI	m/nm	Code	PI	m/nm	Code	Length/Value
User data	nm	193				512 octets maximum

Fig. 28. Encoding of NOT FINISHED SPDU

Notes:

1. The SI field shall have the value 8.
2. The User Data field shall contain any user data supplied by the SS user.

5.3.7 ABORT (AB) SPDU

PGI	m/nm	Code	PI	m/nm	Code	Length/Value
			Transport disconnect	nm	17	1 octet
			Reflect parameter values	nm	49	9 octets maximum
User data	nm	193				9 octets maximum

Fig. 29. Encoding of ABORT SPDU

Notes:

1. The SI field shall contain the value **25.**

2. The Transport Disconnect PV field shall indicate whether or not the transport connection is to be kept, together with an optional reason code. The coding for this field shall be:

   A. bit 1 = 0 ==> transport connection is kept **(never set).**

   B. bit 1 = 1 ==> transport connection released **(always set).**

   C. bit 2 = 1 ==> User abort (see User Data field)

   D. bit 3 = 1 ==> Protocol error

   E. bit 4 = 1 ==> No reason

   F. bits 5-8 reserved (set to zero)

   Bits 5-8 are reserved.

3. The Reflect Parameter Value PV field shall only be present if the Transport Disconnect PV field indicates protocol error, and shall contain an implementation defined value and semantics.

4. The User Data PV field shall only be present if the Transport Disconnect PV field indicates user abort, and shall contain any user data supplied by the SS user.

5.3.8 ABORT ACCEPT (AA) SPDU

Notes:

1. The SI field shall contain the value 26.

2. There is no parameter field associated with this SPDU.

5.3.9 PLEASE TOKENS (PT) SPDU

PGI	m/nm	Code	PI	m/nm	Code	Length/Value
			Token item	nm	16	1 octet
User data	nm	193				512 octets maximum

Fig. 30. Encoding of PLEASE TOKENS SPDU

Notes:

1. The SI field shall contain the value 2.

2. The Token Item PV field shall indicate which tokens are being requested by the sending SS user:

   A. bit 7 = 1 release token

   B. bit 5 = 1 major token

   C. bit 3 = 1 synchronize minor token

   D. bit 1 = 1 data token

   Bits 2, 4, 6 and 8 are reserved.

   Bits corresponding to tokens which are not available are ignored.

3. The User Data PV field shall contain any user data supplied by the SS user. This PGI shall only be present if the Token Item PI unit is present.

This SPDU may be used without the Token Item PI unit and the User Data PGI unit when concatenated with Category 2 SPDUs according to

Bits 2-8 are reserved. If this field is not present, the default shall be as though bit 1 = 1 (i.e. end of SSDU). **This is the case for any OSIRIDE implementation.**

3. The User Information field shall contain any user data supplied by the SS user.

5.3.13 DATA TRANSFER (DT) SPDU

PGI	m/nm	Code	PI	m/nm	Code	Length/Value
			Enclosure Item	nm	25	1 octet
User Information field						Unlimited

Fig. 34. Encoding of DATA TRANSFER

Notes:

1. The SI field shall contain the value 1.
2. There is no parameter field associated with this SPDU.
3. There is a User Information Field associated with this SPDU.

5.3.14 MINOR SYNCHRONIZATION POINT (MIP) SPDU

PGI	m/nm	Code	PI	m/nm	Code	Length/Value
			Sync type Item	m	15	1 octet
			Serial number	m	42	6 octets maximum
User Data	nm	193				512 octets maximum

Fig. 35. Encoding of MINOR SYNC SPDU

Notes:

1. The SI field shall contain the value **49**.

2. The Sync Type Item, if present, shall indicate that the explicit confirmation is not required:

   bit 1 = 1 Confirmation requested

   Bits 2-8 reserved.

   This parameter field shall be absent if an explicit confirmation is required.

3. The Serial Number PV field shall be coded as specified in "5.3.1 CONNECT (CN) SPDU" pag. 56.

4. The User Data PV field shall contain any user data supplied by the SS user.

5.3.15 MINOR SYNC ACK (MIA) SPDU

PGI	m/nm	Code	PI	m/nm	Code	Length/Value
			Serial Number	m	42	6 octets
			User Data	nm	46	512 octets maximum

Fig. 36. Encoding of MINOR SYNC ACK SPDU

Notes:

1. The SI field shall contain the value **50**.
2. The Serial Number PV field shall be coded as specified in "5.3.1 CONNECT (CN) SPDU" pag. 56.
3. The User Data PV field shall contain any user data supplied by the SS user.

5.3.16 MAJOR SYNCHRONIZATION POINT (MAP) SPDU

PGI	m/nm	Code	PI	m/nm	Code	Length/Value
			Sync type Item	m	15	1 octet
			Serial Number	m	42	6 octets maximum
User data	nm	193				512 octets maximum

Fig. 37. Encoding of MAJOR SYNC SPDU

Notes:

1. The SI field shall contain the value **41**.
2. The Sync Type Item shall indicate that this is not the end of an activity:

bit 1 = 1 ==> not end of activity **(always set)**.

Bits 2-8 reserved.

3. The Serial Number PV field shall be coded as specified in "5.3.1 CONNECT (CN) SPDU" pag. 56.

4. The User Data PV field shall contain any user data supplied by the SS user.

5.3.17 MAJOR_SYNC_ACK_(MAA)_SPDU

PGI	m/nm	Code	PI	m/nm	Code	Length/Value
			Serial Number	m	42	6 octets maximum
User data	nm	193				512 octets maximum

Fig. 38. Encoding of MAJOR SYNC ACK SPDU

Notes:

1. The SI field shall contain the value **42**.

2. The Serial Number PV field shall be coded as specified in "5.3.1 CONNECT (CN) SPDU" pag. 56.

3. The User Data PV field shall contain any user data supplied by the SS user.

5.3.18 RESYNCHRONIZE_(RS)_SPDU

PGI	m/nm	Code	PI	m/nm	Code	Length/Value
			Token Setting Item	nm	26	1 octet
			Resync Type	m	27	1 octet
			Serial Number	m	42	6 octets maximum
User data	nm	193				512 octets maximum

Fig. 39. Encoding of RESYNCHRONIZE SPDU

Notes:

1. The SI field shall contain the value 53.

2. The Token Setting Item indicates the requesting SS user's proposed settings for each available token. The bits ans encoding are defined in "5.3.1 CONNECT (CN) SPDU" pag. 56. The values are relevant only if the token is available. If no token is available this parameter need not be present.

3. The Resync Type PV field indicates the resynchronize type which is required:

   A. 0 resynchronize restart

   B. 1 resynchronize abandon

   C. 2 resynchronize set

4. The Serial Number PV field shall be coded as specified in "5.3.1 CONNECT (CN) SPDU" pag. 56.

5. The User Data PV field shall contain any user data supplied by the SS user.

5.3.19 RESYNCHRONIZE ACK (RA) SPDU

PGI	m/nm	Code	PI	m/nm	Code	Length/Value
			Token Setting Item	nm	26	1 octet
			Serial Number	m	42	6 octets maximum
User data	nm	193				512 octets maximum

Fig. 40. Encoding of RESYNCHRONIZE ACK SPDU

Notes:

1. The SI field shall contain the value 34.

2. The Token Setting Item PV field indicates token settings for each token available on the session connection. For the case where the requesting SS user has indicated that the assignment is negotiable ("accepting SS user's choice"), the accepting SS user must fix this assignment. Otherwise, the values in the RESYNCHRONIZE SPDU must be returned.

   This parameter need not be present if no tokens are available on this session connection.

3. The Serial Number PV field shall be coded as specified in "5.3.1 CONNECT (CN) SPDU" pag. 56.

4. The User Data PV field shall contain any user data supplied by the SS user.

Second Part

1.0 INTRODUCTION

This part of the OSIRIDE Session Layer description deals with a complete specification of the implementation aspects.

1.1 SCOPE

The description of the internal structure of the OSIRIDE Session Layer is performed in this second part, followed by the detailed description of each module which constitute it.

1.2 DEFINITIONS

The following list of definitions is only related to this part of the Session Layer document, and refers to the terms and acronyms used in the implementation specifications.

The definitions of the Session Layer (in terms of ISO standards) terms is given in "First Part."

Connection Handler, CH The service task which performs address translations both for incoming and for outcoming connection requests.

Output Queue Handler, OQH The interface that the Session Layer uses to communicate with the external world.

Local Management System, LMS The task, or set of tasks, which performs local management functions. It may require the Session layer to start its activity, to stop it, or information about its behaviour.

Lower Layer Interface, LLI The interface that receives Transport Layer indications.

Scheduler The main task which handles the OSIRIDE Session Layer implementation.

Session Management Functions The functions that the Session Layer offers to the Local Management System.

Session Protocol Machine, SPM. The finite state automata which implements the Session Protocol.

Session User, SU The user of the session layer, i.e. any entity which requests Session services and is not the Local Management System

Upper Layer Interface, ULI The interface that the Session Layer offers to its users.

1.3 FIELD

The field of application of this document is the implementation of an interconnection structure for heterogeneous systems.

2.0 INTERNAL LAYER'S ARCHITECTURE

From a structural point of view, the OSIRIDE Session Layer is made of the modules described in Fig. 41 pag. 80.

Before a detailed description of the functions of each of those modules is given, it is worth to note here some characteristics of the model of the layer, which is depicted in Fig. 41 pag. 80.

A. **The model provides for a unique input queue.**
This means that both messages and signals coming from the Transport Layer or the Upper Layers, and **internal messages** are logically enqueued in the same queue. This has been done in order to simplify the layer's description. It is clear, however, that an implementation may use different mechanisms for external and for internal signals. **The only restriction** is that all control signals are handled by a single entity, which is the **Scheduler**.

B. **The input queue handler** (see also "2.1 Input queue handler" pag. 80) cannot be described in detail, its characteristics being very machine-dependent. It has, however, the only function to dequeue messages from a queue, and to alert the proper task by means of a Scheduler function. All other functions related to upper and lower layer interfaces are handled by specific modules.

C. **The layer management functions** are both internal management functions, like activation/de-activation of the layer, and functions offered to the Session users (see also the ISIDE Access Method [8]).

D. **The output queue handler** cannot be described in detail, its characteristics being very machine-dependent. It has, however, the only function to enqueue messages coming from Session layer functional modules to their appropriate destination, by using Local Operating System's mechanisms.

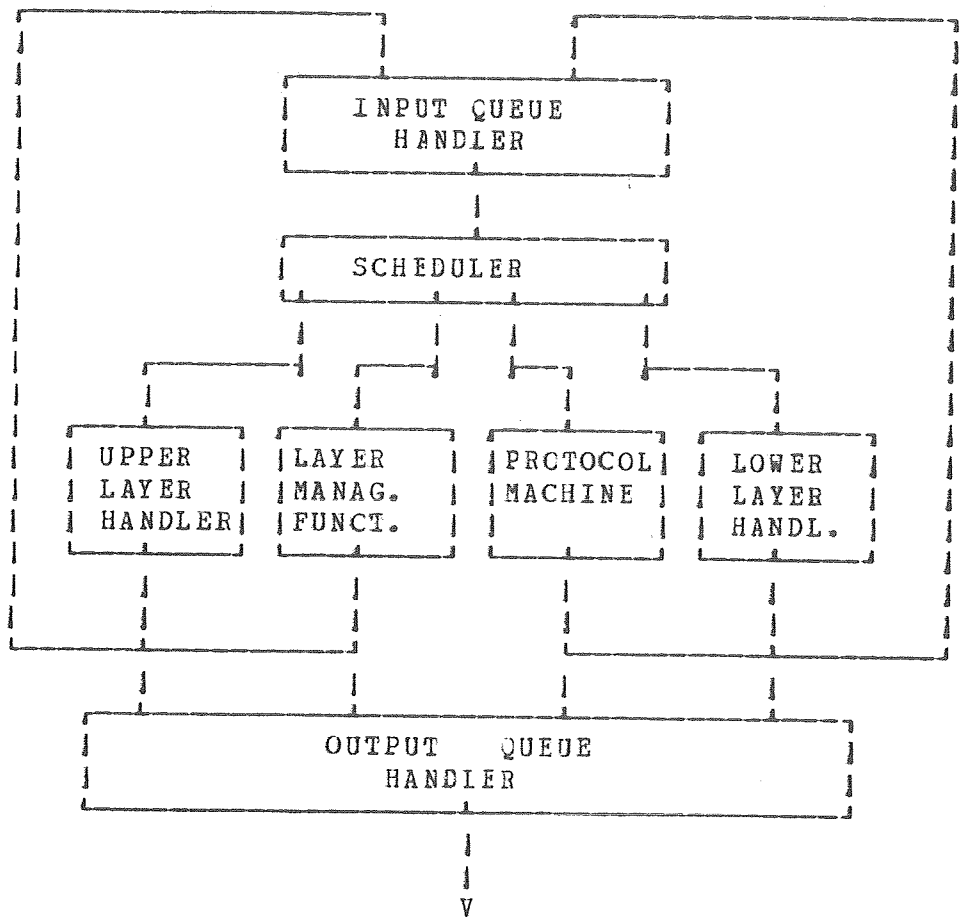


Fig. 41. Internal Layer's structure

2.1 INPUT QUEUE HANDLER

The Input queue handler has the main function of taking requests out of the input queue and passing them to the appropriate modules.

When all requests in the input queue have been dispatched to the appropriate modules, the Input Queue Handler passes control to the Scheduler. It is thus clear that no scheduling functions are performed by the Input Queue Handler, i.e. the modules which the requests are directed to are given control only by the Scheduler.

Requests enqueued are in form of tables which are described in detail in "6.1 Data Structures" pag. 159.

2.2 SCHEDULER

The Session Layer's internal Scheduler is the "dispatcher" of the Session Layer, i.e. that main program which drives all other routines, and which all other routines return to. The Scheduler is **the only part of the Session Layer** which may request such Local Operating System functions as, for instance, **wait:** as a matter of fact, requesting a wait from inside the Session layer means that the entire Session layer is in a waiting state. This may happen only if nothing has to be done, which is an information only known by the Scheduler. The Scheduler is also the "virtual owner" of all the data structures which describe the users' connections. This means that locking mechanisms are used to avoid contemporary access to the same data structure by more than one different modules, and those mechanisms are internal to the Scheduler. On the other hand, data structures may be created and destroyed by other modules than the Scheduler. For a description of the data structures used by the OSIRIDE Session layer, see "6.1 Data Structures" pag. 159.

Note: Any module herein described, when accessing a data structure, issues a "lock" operation on it, to prevent contemporary access by other modules. An "unlock" operation is then issued, as soon as there is no more need of the data structure. This **does not mean** that lock and unlock mechanisms must be implemented, but rather that, if a particular implementation is so structured as to require locks, that is the place in the program where such functions should be invoked. Any module is called by the Scheduler and, as soon as it finishes its functions, returns to the Scheduler.

2.3 PROTOCOL MACHINE

The protocol machine task is the implementation of the protocol described in "First Part." This implementation is described firstly as a set of state tables, then as a set of routines which represent the intersections states-events in the state tables. The detailed implementation description of the OSIRIDE Session layer protocol machine is given in "4.0 Protocol state machine" pag. 116.

The protocol machine is driven (by means of Scheduler functions) by:

The upper layer handler

The lower layer handler

and, on its turn, drives (by means of Scheduler functions):

The output queue handler

2.4 LOWER LAYER INTERFACE HANDLER

The Lower layer interface handler is based upon the interface provided by the OSIRIDE Transport Layer. Its main features are described in "3.0 Interface logical structure" pag. 85, whilst the detailed implementation description is given in "6.5 Lower layer interface" pag. 229.

From now on, the acronym LLI may be used instead of Lower Layer Interface.

The LLI is driven (by means of Scheduler functions) by:

The Input queue handler

and, on its turn, drives (by means of Scheduler functions):

The protocol machine

The output queue handler

2.5 UPPER LAYER INTERFACE HANDLER

The Upper layer interface handler provides for the user requests handling (i.e. ISIDE). Its main features are described in "3.0 Interface logical structure" pag. 85, whilst the detailed implementation description is given in "6.3 Upper layer interface" pag. 164.

From now on, the acronym ULI may be used instead of Upper Layer Interface.

The ULI is driven (by means of Scheduler functions) by:

The Input queue handler

and, on its turn, drives (by means of Scheduler functions):

The protocol machine

The output queue handler

2.6 SESSION MANAGEMENT FUNCTIONS

The Session management functions are those functions that the Session layer offers both to the Local Management System and to the users. As a first implementation, the following functions have been identified:

- **Session layer activation and de-activation.**
  The Local Management System activates the Session Layer by means of an Activate command. The Session Layer sets its Global State internal variable to **ACTIVE**. Other values of the Global State variable are:

  - **OFF**, which is the state after initialization and before receiving an Activate command. In this state, only Activate and de-Activate commands are accepted.

  - **FLUSHING**, which is the state after a Flush command has been accepted. No new Session connections may be requested in this state, and all incoming connection requests are refused. As soon as all already existing Session connections are terminated, the closing procedure is activated, and the Global State variable is set to **off**.

  - **CLOSING**, which is the state after a de-Activate command is received. No commands are accepted in this state, which terminates when all internal termination functions are terminated. At that point, the value of the Global State variable is **OFF**.

- **Request of information**
  This function is requested by the Local Management System when an ISIDE STATUS request is received. Information is transferred to the LMS without any security check, which is supposedly performed by the LMS itself.

The implementation specifications of the Session Layer Management interface are described in "6.6 Session management functions" pag. 235, while "5.0 Management Functions" pag. 155 deals with the valid events and the related actions which are taken by the Session Layer Management functions.

The Layer Manager is driven (by means of Scheduler functions) by:

The Input queue handler

and, on its turn, drives (by means of Scheduler functions):

The output queue handler

2.7 OUTPUT QUEUE HANDLER

The Output Queue Handler (OQH) functions are those of communicating signals and data to the Session users or to the OSIRIDE Local Management System. No detailed description can be done for this module, its behaviour being stricly related to the Host Operating System Inter-task communication facilities.

The OQH is driven (by means of Scheduler functions) by:

The Upper layer handler

The Lower layer handler

The Protocol machine

The Layer Management Functions

and, on its turn, transfers messages and signals to either the Session users or to the OSIRIDE Local Management System.

The OQH implements the message transfer by means of the **MESSAGE** primitive, which is used by all other modules which intend to use the OQH.

3.0 INTERFACE LOGICAL STRUCTURE

Two main interfaces are defined inside the OSIRIDE Session Layer, which are: the **Upper Layer Interface** and the **Lower Layer Interface.** Both are driven by the **Scheduler**, when events are recognized by the **Input queue handler**.

3.1 UPPER LAYER INTERFACE

The Upper Layer Interface recognizes events coming from the entities which represent the Session users, i.e. the requests coming from **ISIDE.**

According to the operations which are requested, the ULI may be able to answer (negatively or positively) to the requestor. There are cases, however (typically when the Protocol Machine is involved), when the ULI does not answer directly to the requestor, but generates an internal event for another Session Layer module. In these cases, the invoked module will answer to the originator of the request.

In what follows, the valid events for the Upper Layer Interface are described in detail, with related parameters, together with the events generated by the Upper Layer Interface towards both the user and other functional modules of the Session Layer.

3.1.1 OPEN event

OPEN is a user initiated event, which states the willingness of the user to be known by the OSIRIDE software.

3.1.1.1 Parameters

What follows is the list of the parameters related to the OPEN event:

- **name**
- **password**
- **maxout**
- **maxin**

3.1.1.2 Procedure

At reception of an OPEN event, the ULI tries to allocate a User Control Block (UCB, see also "6.1 Data Structures" pag. 159). If this is not possible, a MESSAGE is generated towards that specific user, with reason code = No Core Available.

If the Global State variable is set to **FLUSHING**, then a MESSAGE is generated, with reason code = Session Layer Terminating.

The User Control Block is filled in with:

name, which is used afterwards to identify that user

password, which is used afterwards to check for the user's right to ask for certain services.

maxout, which is used afterwards to check that the user does not try to issue more than <maxout> CONNECT requests.

maxin, which is used afterwards by the Lower Layer Interface to automatically refuse a CONNECT indication, if the number of already active incoming connection is <maxin>.

After that, the user is notified by means of a MESSAGE, with parameter = <OSIRIDE name of the local system>. This is a variable which is known by the Session Layer, and may be useful for an application program.

A MESSAGE is then sent to the LMS, with parameter = Open from **userid** with **name**, where:

userid is the local identifier of the user who issued the OPEN request, and

name is the identifier specified in the request.

3.1.2 CLOSE event

CLOSE is a user initiated event, which states the willingness of the user to be detached from the OSIRIDE software.

3.1.2.1 Parameters

There is only one parameter related to the CLOSE event, which is:

- **name**

3.1.2.2 Procedure

At reception of a CLOSE event, the ULI tries to find a UCB whose **name** field matches with the parameter passed with the CLOSE command. Then:

- If no UCB is found with the specified name, a MESSAGE is generated with reason code = Cpen not issued.

- Otherwise, it is checked whether the CLOSE request arrived from the same user whose connections have to be closed, or from the Local Management System.

  - If either condition is not true, the ULI generates a MESSAGE to the requestor, with reason code = Not Authorized.

  - Otherwise, the ULI:

    — Locks the UCB related to that user name.

    — Generates an ABORT.request on any active Session connection (identified by means of Communication Control Blocks, CCB, see also "6.1 Data Structures" pag. 159) related to that user name towards the Session Protocol Machine.

    — Freezes all the Connection Control Blocks (CCB) related to that user, chaining them to the "Frozen CCBs Queue" (FCQ, see also "6.1 Data Structures" pag. 159).

    — Notifies the user by means of a MESSAGE (without parameters).

    — Notifies the LMS by means of a MESSAGE, with parameter = User **name** is closing, where:

      name is the parameter of the CLOSE request.

    — Frees the User Control Block (UCB, see also "6.1 Data Structures" pag. 159), which was created at OPEN time.

    — Decrements the Global Users Number variable by one.

    — If the Global Users Number is equal to zero, then:

      - If the Global State variable is set to **FLUSHING**, then its value is changed to **CLOSING**, and the closing procedures are initiated.

3.1.3 READY event

READY is a user initiated event, which states the willingness of the user to accept incoming CONNECT requests.

3.1.3.1 Parameters

What follows is the list of the parameters related to the READY event:

- **system**
- **user**
- **name**
- **password**

3.1.3.2 Procedure

The UCB related to that user's name is searched. Then:

- If the UCB does not exist, a MESSAGE is generated, with reason code = Cpen not issued.
- If the UCB does exist, then:
  - the ULI locks the UCB related to that user name, then the <maxin> value is checked:
    - If <maxin> = 0, a MESSAGE is initiated towards the user who issued the READY request.
    - If <maxin> > 0, then:
      - The READY field of the UCB is set to **yes** (see also "6.1 Data Structures" pag. 159).
      - The **SYSTEM** and/or the **USER** and/or the **PASSWORD** fields of the UCB are set to the specified values, if any of them has been specified.
      - A MESSAGE is delivered to the user who issued the request.
  - The UCB is unlocked.

3.1.4 RECEIVE event

RECEIVE is a user initiated event, which states the willingness of the user to accept incoming messages coming from the partner.

3.1.4.1 Parameters

There are two parameters related to the RECEIVE event, which are:

- **connection id**
- **name**

3.1.4.2 Procedure

At reception of a RECEIVE event, a UCB is searched which matches the specified user's name. Then:

- If the UCB does not exist, a MESSAGE is generated, with reason code = Open not issued.
- If the UCB does exist, then a CCB chained to that UCB is searched which contains a **connection id** field with the same value as specified. Then:
  - If no CCB is found, a MESSAGE is generated, with reason code = Invalid Request.
  - If either the **OUTFIN** or the **ABORT** fields of the CCB bring the value **yes**:
    - A MESSAGE is generated, with parameter = Connection closing.

    Otherwise:

    - The CCB is locked.
    - If something is available for that user, a MESSAGE is generated towards that user, which contains the information to be passed. The data buffer containing the user information is then released, and a new block is allocated (of different length), based on the need of the Transport layer. This block is then passed to the Transport Layer by means of an Enable Reception interface command. If the allocation of the new block is not possible, due for instance to temporary memory shortage, nothing is done. This means that, if the user does not issue RECEIVE, the connection will probably be blocked by back pressure.
    - If nothing is available at the moment, the **RECEIVE** field of the CCB (see also "6.1 Data Structures" pag.

159) is set to **yes**, and a MESSAGE is generated, with parameter = No Data.

— The CCB is unlocked.

3.1.5 RESET event

The RESET event indicates the user's willingness to reset a previously issued operation.

3.1.5.1 Parameters

There are three parameters related to the RESET event:

- **operation**
- **connection id**
- **name**

3.1.5.2 Procedure

At reception of a RESET event, a UCB is searched which matches the specified user's name. Then:

- If the UCB does not exist, a MESSAGE is generated, with reason code = Open not issued.
- If the UCB does exist, then the **operation** parameter is checked:
  - If <operation> <> RECEIVE or READY, the ULI generates a REJECT event towards that user, with parameter = Invalid Request.
  - If <operation> = RECEIVE, a CCB is searched which contains a **connection id** field which matches the specified value. Then:
    - If no CCB is found, a MESSAGE is generated, with parameter = Invalid Connection id.
    - If the CCB is valid, then:
      - The CCB is locked.
      - The field RECEIVE in the CCB is set to **no**.
      - A MESSAGE is generated towards that user with no parameters.
      - The CCB is unlocked.

- If <operation> = READY, then:
  - The UCB is locked
  - The READY field of the UCB is set to **no**.
  - The **SYSTEM, USER** and **PASSWORD** fields of the UCB are set to binary zeroes.
  - A MESSAGE is generated towards that user with no parameters.
  - The UCB is unlocked.

3.1.6 CONNECT_event

The CONNECT event indicates the user's willingness to establish a connection with a remote counterpart. This event drives the Session Protocol machine.

3.1.6.1 Parameters

There are nine parameters related to the CONNECT event:

- **system**
- **user**
- **profile**
- **data token**
- **mark token**
- **sync token**
- **term token**
- **password**
- **name**

3.1.6.2 Procedure

At reception of a CONNECT event, a UCB is searched which matches the specified user's name. Then:

- If the Global State variable is set to **FLUSHING**, a MESSAGE is generated, with reason code = Session Layer Terminating.

- If the UCB does not exist, a MESSAGE is generated, with reason code = Open not issued.

- If the UCB does exist, then the **system** parameter is checked:

  - If <system> is not an already known system name, the ULI generates a REJECT event towards that user, with parameter = Invalid System.

  - If <system> is valid, the **profile** parameter is checked. Then:

    — If the **profile** is invalid, a MESSAGE is generated, with reason code = Invalid Profile.

    — If the password field of the UCB does not match with the password requested for that profile, a MESSAGE is generated, with parameter = Profile not Authorized.

    — If the field **ACTOUT** of the UCB is already equal to the field **maxout**, a MESSAGE is generated, with parameter = Too many connections out.

    — Otherwise:

      - The UCB is locked.

      - A CCB is created, and chained to the CCBs chain of the UCB.

      - The field **SYSTEM** in the CCB is set to the user's specified value.

      - The field **USER** in the CCB is set to the user's specified value.

      - If the specified user's profile requires concatenation, then the field **CONCATENATION** in the CCB is set to **yes**, otherwise it is set to **no**.

      - The field **ACTOUT** of the UCB is incremented by one.

      - A CONNECT.request is prepared for the SPM, with the **data, mark, sync** and **term** tokens parameters set as specified by the user. The **password** parameter, if specified, is delivered as **user data**. The selected profile is passed to the SPM, in order for it to request the appropriate Transport connection characteristics.

      - A MESSAGE is generated towards the user with parameter = Connection id.

      - A MESSAGE is generated towards the LMS with parameter = User name connecting with **USER** at **SYSTEM**, where:

name is the local identifier of the user

USER is the remote user identifier specified in the CONNECT request.

SYSTEM is the identifier of the remote system.

- The UCB is unlocked.

3.1.7 RESPONSE event

The RESPONSE event indicates the user's willingness to answer to a previously received indication which has to be confirmed.

3.1.7.1 Parameters

The parameters of a RESPONSE event vary according to the operation which is actually requested. The following is the list of all possible parameters.

- **name**
- **connection id**
- **operation**
- **concatenation**
- **data token**
- **mark token**
- **sync token**
- **term token**
- **point**
- **type**
- **result**
- **mark-data**
- **tok-data**
- **res-data**
- **data**

3.1.7.2 Procedure

At reception of a RESPONSE event, a UCB is searched which matches the specified user's name. Then:

- If the UCB does not exist, a MESSAGE is generated, with reason code = Open not issued.

- If the UCB does exist, then a CCB is searched which bears in the **connection id** field the same value as that specified.

  - If no CCB is found, a MESSAGE is generated, with parameter = Invalid Connection id.

  - Otherwise:

    — The CCB is locked.

    — The **operation** parameter is checked:

      - If <operation> = **CONNECT**, the following parameters are examined:

        result

        data token

        mark token

        sync token

        term token

        concatenation

        data

        Then:

        * If length of **data** > 512 octets, a MESSAGE is generated, with parameter = Incorrect data length.

        * If <response> = **negative**, then:

          - A CONNECT.confirm is generated towards the SPM, with value = **no** and the <data> parameter is to be passed as user data.

          - The CCB related to that connection is "frozen", as it is chained to the Frozen CCBs Queue.

          - The user is notified by means of a MESSAGE.

        * If <response> = **positive**, then:

yes.

- A CONNECT.confirm is generated towards the SPM, with value = **yes** and the <data> parameter is to be passed as user data.

- The value of the **ACTIM** field in the CCB is incremented by one.

- The user is notified by means of a MESSAGE with no parameters.

- A MESSAGE is generated towards the LMS with parameter = User **name** connected with **USER** at **SYSTEM**, where:

  name is the local identifier of the user

  USER is the remote user identifier specified in the CONNECT request.

  SYSTEM is the identifier of the remote system.

* If <operation> = **SYNC**, the following parameters are examined:

  type

  point

  mark-data

  Then the length of mark-data parameter is examined:

  * If length > 512 octets, a MESSAGE is generated, with parameter = Incorrect Data Length.

  * If length ≤ 512 octets, the **point** parameter is examined:

    - If <point> is not a valid synchronization point, a MESSAGE is generated, with parameter = Wrong synchronization point.

    - If <point> is valid:

      • The **type** parameter is checked:

        — If type = **Major**, a MAJOR_SYNC confirmation is generated towards the SPM, and the **data** parameter is to be passed as user data.

— If type = **Minor**, a MINOR_SYNC confirmation is generated towards the SPM, and the **data** parameter is to be passed as user data.

- A MESSAGE is generated towards the user, with no parameters.

- If <operation> = **SYNC+PLEASE**, then the following parameters are examined:

  mark-data

  tok-data

  point

  type

  data token

  mark token

  sync token

  term token

  Then:

  * The lengths of mark-data and tok-data are checked:

    - If length > 512 octets, a MESSAGE is generated towards the user, with parameter = Incorrect Data Length.

    - If length ≤ 512 octets, then the **point** parameter is examined:

      - If **point** is not a valid synchronization point, a MESSAGE is issued, with parameter = Wrong sync point.

      - If **point** is valid:

        — The field **CONC** of the CCB is set to **yes**

        — The **type** parameter is checked:

          - If <type> = **minor**, then:

            * A MINOR_SYNC.response is generated for the SPM, with the serial number parameter set at the **point** value.

- If <type> = **major**, then:
  - A MAJOR_SYNC.response is generated for the SPM, with the serial number parameter set at the **point** value.
- Data contained in mark-data are passed as user data field parameter.
- The **DELIVER** field of the CCB is set to **yes**.
- A PLEASE_TOKENS.request is generated towards the SPM, which brings the tokens to be requested.
- Data contained in tok-data are passed as data field parameter.
- The CONC field of the CCB is set to **no**.
- The DELIVER field of the CCB is set to **no**.

- If <operation> = **SYNC+GIVE**, then the following parameters are examined:

  mark-data

  tok-data

  point

  type

  data token

  mark token

  sync token

  term token

  Then the **CONCATENATION** field of the CCB is checked:

  * If <CONCATENATION> = **no**, a MESSAGE is generated towards the user, with parameter = Concatenation not allowed.
  * If <CONCATENATION> = **yes**, then:
    - The lengths of mark-data and tok-data are checked:

— If length > 512 octets, a MESSAGE is generated towards the user, with parameter = Incorrect Data Length. The field **CONC** is set to **no**.

— If length ≤ 512 octets, then the **point** parameter is checked:

- If **point** is not a valid synchronization point, a MESSAGE is generated, with parameter = Wrong sync point.
- If **point** is valid, then:
  - * The field **CONC** of the CCB is set to **yes**
  - * The **type** parameter is checked:
    - If <type> = **minor**, then:
      - A MINOR_SYNC response is generated for the SPM, with the serial number parameter set at the **point** value.
    - If <type> = **major**, then:
      - A MAJOR_SYNC response is generated for the SPM, with the serial number parameter set at the **point** value.
  - * Data contained in mark-data are passed as user data field parameter.
  - * The **DELIVER** field of the CCB is set to **yes**.
  - * A GIVE_TOKENS.request is generated towards the SPM, which brings the tokens to be GIVEn.

* Data contained in tok-data are passed as user data field parameter.

* The CONC field of the CCB is set to **no**.

* The DELIVER field of the CCB is set to **no**.

* If <operation> = **RESYNC**, the following parameters are examined:

  point

  result

  type

  data token

  mark token

  sync token

  term token

  res-data

  Then:

  * The **point** parameter is examined. Then:

    - If point is not a valid resynchronization point, a MESSAGE is generated, with parameter = Wrong resync point.

    - If point is valid, the length of **res-data** parameter is checked:

      • If length > 512 octets, a MESSAGE is generated, with parameter = Incorrect Data Length.

      • If length ≤ 512 octets:

        — A RESYNC.confirmation is generated towards the SPM, with **type** and **point** values equal to the corresponding parameters, **data**, **mark**, **sync** and **term** tokens parameters passed as new positions of the related tokens, if they have been specified, and the **data** parameter passed as user data.

— A MESSAGE is generated towards the user, with no parameters.

* If <operation> = **FINISH**, the following parameters are examined:

result

data

Then the length of **data** parameter is checked:

- If length > 512 octets, a MESSAGE is generated, with parameter = Incorrect Data length.
- If length ≤ 512 octets, the **result** parameter is checked:
  - If <result> = **reject**, then:
    — A RELEASE(negative).response is generated for the SPM.
  - If <result> = **accept**, then:
    — A RELEASE(positive).response is generated for the SPM.

— The CCB is unlocked.

3.1.8 TRANSFER event

The TRANSFER event indicates the user's willingness to transfer information to the remote counterpart. This may be either data or control information. This event drives the Session Protocol machine.

3.1.8.1 Parameters

The parameters of a TRANSFER event vary according to the operation which is actually requested. The following is the list of all possible parameters.

- **name**
- **connection id**
- **operation**
- **data token**

- **mark token**
- **sync token**
- **term token**
- **point**
- **type**
- **data**
- **tok-data**
- **mark-data**

3.1.8.2 Procedure

At reception of a TRANSFER event, a UCB is searched which matches the specified user's name. Then:

- If the UCB does not exist, a MESSAGE is generated, with reason code = Open not issued.
- If the UCB does exist, then a CCB is searched which bears in the **connection id** field the same value as that specified:
  - If no CCB is found, a MESSAGE is generated, with parameter = Invalid Connection id.
  - Otherwise:
    - The CCB is locked.
    - The **operation** parameter is checked:
      - If <operation> = **DATA**, the only valid parameter is:

        data

        Then:

        * Data are transmitted by means of a DATA.request to the SPM.
      - If <operation> = **PLEASE**, the following parameters are examined:

        data token

        mark token

        sync token

        term token

Then:

* A PLEASE_TOKENS.request is generated towards the SPM, which brings the tokens to be GIVEn.

- If <operation> = **GIVE**, the following parameters are examined:

  data token

  mark token

  sync token

  term token

  tok-data

  Then the length of tok-data is checked:

  * If length > 512 octets, a MESSAGE is generated towards the user, with parameter = Incorrect Data Length.

  * If length ≤ 512 octets, then:

    - A GIVE_TOKENS.request is generated towards the SPM, which brings the tokens to be GIVEn. Data contained in tok-data are passed as data field parameter.

- If <operation> = **SYNC**, the following parameters are examined:

  point

  type

  mark-data

  The length of mark-data is checked:

  * If length > 512 octets, a MESSAGE is generated towards the user, with parameter = Incorrect Data Length.

  * If length ≤ 512 octets, then the **type** parameter is checked:

    - If <type> = **minor**, then:

      - A MINOR_SYNC.request is generated for the SPM, with the serial number parameter set at the **point** value.

    - If <type> = **major**, then:

- A MAJOR_SYNC.request is generated for the SPM, with the serial number parameter set at the **point** value.

- Data contained in mark-data are passed as user data field parameter.

- If <operation> = **RESYNCH**, then the following parameters are examined:

  point

  type

  res-data

  Then the length of **res-data** is checked:

  * If length > 512 octets, a MESSAGE is generated, with parameter = Incorrect Data Length.

  * If length ≤ 512 octets, the **point** parameter is checked:

    - If point is not a valid resynchronization point, a MESSAGE is generated, with parameter = Wrong resync point.

    - If point is valid:

      - A RESYNCHRONIZE.request is passed to the SPM with **restart** or **abandon** or **set** value according to <type>.

      - Res-data is passed as user data field.

- If <operation> = **DATA+GIVE**, then the following parameters are examined:

  data

  tok-data

  data-token

  mark-token

  sync-token

  term-token

  Then:

  * The length of tok-data is checked:

- If length > 512 octets, a MESSAGE is generated towards the user, with parameter = Incorrect Data Length.

- If length ≤ 512 octets, then:

  - The field **CONC** of the CCB is set to **yes**

  - A DATA.request is generated for the SPM.

  - The **DELIVER** field of the CCB is set to **yes**.

  - A GIVE_TOKENS.request is generated for the SPM.

  - Data contained in tok-data are passed as user data field parameter.

  - The CONC field of the CCB is set to **no**.

  - The DELIVER field of the CCB is set to **no**.

- If <operation> = **DATA+SYNC**, then the following parameters are examined:

  data

  mark-data

  point

  type

  Then the **CONCATENATION** field of the CCB is checked:

  * If <CONCATENATION> = **no**, a MESSAGE is generated towards the user, with parameter = Concatenation not allowed.

  * If <CONCATENATION> = **yes**, then:

    - The length of mark-data is checked:

      - If length > 512 octets, a MESSAGE is generated towards the user, with parameter = Incorrect Data Length.

      - If length ≤ 512 octets, then the **point** parameter is checked:

        — If **point** is not a valid synchronization point, a MESSAGE is gener-

ated, with parameter = Wrong sync point.

— If point is valid, then:

- The field **CONC** of the CCB is set to **yes**
- A DATA.request is generated for the SPM.
- The **DELIVER** field of the CCB is set to **yes**.
- The **type** parameter is checked:
  - If <type> = **minor**, then:
    - A MINOR_SYNC.request is generated for the SPM, with the serial number parameter set at the **point** value.
  - If <type> = **major**, then:
    - A MAJOR_SYNC.request is generated for the SPM, with the serial number parameter set at the **point** value.
- Data contained in mark-data are passed as user data field parameter.
- The CONC field of the CCB is set to **no**.
- The DELIVER field of the CCB is set to **no**.

- If <operation> = **SYNC+GIVE**, then the following parameters are examined:

  mark-data

  tok-data

  point

  type

  data token

  mark token

sync token

term token

Then the lengths of mark-data and tok-data are checked:

* If length > 512 octets, a MESSAGE is generated towards the user, with parameter = Incorrect Data Length.

* If length ≤ 512 octets, the **point** parameter is checked:

  - If **point** is not a valid synchronization point, a MESSAGE is generated, with parameter = Wrong sync point.

  - If **point** is valid, then:

    • The field **CONC** of the CCB is set to **yes**

    • The **type** parameter is checked:

      — If <type> = **minor**, then:

        ▪ A MINOR_SYNC.request is generated for the SPM, with the serial number parameter set at the **point** value.

      — If <type> = **major**, then:

        ▪ A MAJOR_SYNC.request is generated for the SPM, with the serial number parameter set at the **point** value.

    • Data contained in mark-data are passed as user data field parameter.

    • The **DELIVER** field of the CCB is set to **yes**.

    • A GIVE_TOKENS.request is generated towards the SPM, which brings the tokens to be GIVEn.

    • Data contained in tok-data are passed as data field parameter.

    • The CONC field of the CCB is set to **no**.

    • The DELIVER field of the CCB is set to **no**.

- If <operation> = **DATA+SYNC+GIVE**, then the following parameters are examined:

  data

  mark-data

  tok-data

  point

  type

  data token

  mark token

  sync token

  term token

  Then the **CONCATENATION** field of the CCB is checked:

  * If <CONCATENATION> = **no**, a MESSAGE is generated towards the user, with parameter = Concatenation not allowed.

  * If <CONCATENATION> = **yes**, then:

    - The lengths of mark-data and tok-data are checked:

      - If length > 512 octets, a MESSAGE is generated towards the user, with parameter = Incorrect Data Length.

      - If length ≤ 512 octets, the **point** parameter is checked:

        — If **point** is not a valid synchronization point, a MESSAGE is generated, with parameter = Wrong sync point.

        — If **point** is valid, then:

          - The field **CONC** of the CCB is set to **yes**

          - A DATA.request is generated for the SPM.

          - The **type** parameter is checked:

            * If <type> = **minor**, then:

- A MINOR_SYNC.request is generated for the SPM, with the serial number parameter set at the **point** value.

* If <type> = **major**, then:

- A MAJOR_SYNC.request is generated for the SPM, with the serial number parameter set at the **point** value.

- Data contained in mark-data are passed as user data field parameter.
- The **DELIVER** field of the CCB is set to **yes**.
- A GIVE_TOKENS.request is generated towards the SPM, which brings the tokens to be GIVEn.
- Data contained in tok-data are passed as data field parameter.
- The CONC field of the CCB is set to **no**.
- The DELIVER field of the CCB is set to **no**.

3.1.9 ABORT_event

The ABORT event indicates the user's willingness to abruptly terminate a connection with a remote counterpart. This event drives the Session Protocol machine.

3.1.9.1 Parameters

There are two parameters related to the ABORT event:

- **name**
- **connection id**
- **reason**

3.1.9.2 Procedure

At reception of an ABORT event, a UCB is searched which matches the specified user's name. Then:

- If the UCB does not exist, a MESSAGE is generated, with reason code = Open not issued.
- If the UCB does exist, then a CCB is searched, which brings the specified **connection id**. Then:
  - If no CCB is found, a MESSAGE is generated, with reason code = Invalid connection id
  - If a CCB is found, then:
    - The CCB is locked.
    - The **ABORT** field of the CCB is set to **yes**.
    - An ABORT.request is generated for the SPM, which brings, as reason code, the **reason** parameter.
    - A MESSAGE is generated towards the user, with no parameter.
    - The CCB is unlocked.

3.1.10 FINISH event

The FINISH event indicates the user's willingness to softly terminate a connection with a remote counterpart. This event drives the Session Protocol machine.

3.1.10.1 Parameters

There are two parameters related to the FINISH event:

- **name**
- **connection id**

3.1.10.2 Procedure

At reception of a FINISH event, a UCB is searched which matches the specified user's name. Then:

- If the UCB does not exist, a MESSAGE is generated, with reason code = Open not issued.

- If the UCB does exist, then a CCB is searched, which brings the specified **connection id**. Then:

  - If no CCB is found, then:

    — A MESSAGE is generated, with reason code = Invalid connection id

  - Otherwise:

    — The CCB is locked.

    — The **OUTFIN** field of the CCB is set to **yes**.

    — A RELEASE.request is generated for the SPM.

    — The CCB is unlocked.

3.2 LOWER LAYER INTERFACE

The Lower Layer Interface recognizes events coming from only one entity, the Transport Layer. Just as for the Upper Layer Interface, the Lower Layer Interface (hereafter shortly called LLI) may either directly answer (negatively or positively) to the requestor, or generate an internal event for another Session Layer module (the Session protocol machine). In this latter case, the other module itself will answer to the request.

In what follows, the valid events for the LLI are described in detail, together with the events generated by the LLI towards the Transport layer, other functional modules of the Session layer or the Local Management System.

3.2.1 T-connect.indication event

T-connect.indication is an event coming from the Transport layer, which states the willingness of a remote Session entity to establish a Transport connection.

3.2.1.1 Parameters

What follows is the list of the parameters related to the T-connect.indication event:

- **Reference number**
- QoS

- **Remote Transport address**

3.2.1.2 Procedure

At the reception of a T-connect.indication event, the LLI checks the Global State Variable.

- If this variable is not set to **Active:**
  - a T-disconnect.request is generated with

    Reference number as carried by the T-connect.indication

    reason = Session Layer Terminating

    no tcep-identifier (as not yet assigned).
- Otherwise the number of existing CCBs is checked:
  - If the number of the existing CCBs is less than the max number of possible CCBs[5]:
    - A CCB is created and chained to the Incoming Requests chain.
    - The **Reference Number** field of the CCB is filled with the reference number carried by the T-connect.indication.
    - a Tcep-identifier is assigned to the CCB and the **TCEP** field of the CCB is filled with this value[6].
    - The CCBSTATE is set to STA1C.
    - A block of 650 octets is requested and chained to the CCB, in order to receive the CONNECT message.
    - A T-connect.response is generated with:

      Reference Number as that carried by the T-connect.indication

      Tcep-identifier as assigned to this CCB

      local Transport address **(always 1 (one) in the first OSIRIDE implementation)**

---

5 This number is a **pure local choice, and may be even undefined.**

6 No details are given on the assignement mechanism of the Tcep-id, which is a local implementation choice. The only constraint here given is that **the Tcep-id should be unique in the Session layer, i.e. two different connections, even of the same Session user, will have different Tcep-ids.**

pointer and length of the buffer which contains the initial CONNECT SPDU (650 octets is the assigned length).

QoS as received in the T-connect.indication

- If one of the above listed functions cannot be performed, due for instance to lack of memory, a T-disconnect.request is generated with:

  Reference Number, as carried by the T-connect.indication

  No Tcep-id (not yet allocated)

  reason = no memory available

3.2.2 T-connect.confirm event

T-connect.confirm is an event coming from the Transport layer, which states the positive response to a previous T-connect.request. This command informs the Session layer that the requested Transport connection has been established.

3.2.2.1 Parameters

What follows is the list of the parameters related to the T-connect.confirm event:

- Reference number
- QoS
- Tcep-identifier

3.2.2.2 Procedure

At the reception of a T-connect.confirm event, a CCB is searched with the Tcep-identifier carried by the T-connect.confirm event, and the CCBSTATE field of the CCB is checked.

- If CCBSTATE is STA1B (waiting for confirm), then the first T-DATA.request is generated for this connection, carrying the CONNECT SPDU chained to the CCBTDATA chain of the CCB.
- Otherwise, a T-disconnect.request is generated with:

  Tcep-identifier, as carried by the T-connect.confirm

  Reference Number, as carried by the T-connect.confirm

reason = Tcep-id invalid

3.2.3 T-data.indication event

T-data.indication is an event coming from the Transport layer, which indicates the delivery of a TSDU. This command informs the Session layer of the identifier of the connection, and of the address of a memory block containing the data.

3.2.3.1 Parameters

What follows is the list of the parameters related to the T-data.indication event:

- **Reference number**
- **TSDU pointer**
- **TSDU length**

3.2.3.2 Procedure

At the reception of a T-data.indication event, a CCB is searched carrying the specified Tcep-identifier.

- If no CCB is found, the Local Management will be informed by means of a MESSAGE request.
- Otherwise, the LLI:
  - decrements the **CCBTDATA_AVAILABLE** CCB field by the length of data carried by the T-data.indication. This is done in order for the Session layer to know how much memory is needed by the Transport layer for that connection, tc allocate a corresponding buffer as soon as this data buffer will be de-allocated. The new allocation is made to restore the initial buffer assignement which is made for each connection at CONNECT time.
  - If SPDUs are concatenated in the TSDU, they are separated and passed one a time to the SPM routines. The rules for chaining and de-chaining are dealt with in "3.4.4 Concatenation" pag. 25 in "First Part."

3.2.4 T-expedited-data.indication event

This indicates the delivery of an expedited TSDU. In the message, the Transport layer specifies the identifiers of the connection and the expedited (maximum size = 16 octets).

3.2.4.1 Parameters

What follows is the list of the parameters related to the T-ex-data.indication event:

- **Reference number**
- **Ex-data (max 16 octets)**

3.2.4.2 Procedure

At the reception of a T-ex-data.indication event, a CCB is searched carrying the specified Tcep-identifier.

- If no CCB is found, the Local Management will be informed by means of a MESSAGE request.
- Otherwise, the LLI:
  - decodes the received Protocol Data Unit
  - calls the SPM.

3.2.5 T-disconnect.indication event

This indicates the refusal or clear of a Transport connection.

3.2.5.1 Parameters

What follows is the list of the parameters related to the T-disconnect.indication event:

- **Reference number**
- **reason code.**

3.2.5.2 Procedure

At the reception of a T-disconnect.indication event, a CCB is searched carrying the specified Tcep-identifier.

- If no CCB is found, the Local Management will be informed by means of a MESSAGE request.

- Otherwise, the LLI checks the CCBSTATE field:

  - If the CCBSTATE field has a value different from STA1 or STA16, this means that the disconnect is a Transport generated event (for instance, a network failure), and an ABORT SPDU is created and chained to the INSPDU chain of the related CCB.

  - If the CCBSTATE field has the value STA1 or STA16, this means that the disconnect is a peer entity generated event. The CCB should be already chained to the "frozen CCB" queue. The CCB is de-allocated.

3.2.6 Enable-transmission event

This indicates that a complete TSDU has been transmitted. The Transport layer gives back the memory area that contained the TSDU.

3.2.6.1 Parameters

What follows is the list of the parameters related to the Enable-transmission event:

- **Reference number**

- **TSDU pointer**

- **TSDU length**

3.2.6.2 Procedure

At the reception of an Enable-transmission event, a CCB is searched carrying the specified Tcep-identifier.

- If no CCB is found, the Local Management will be informed by means of a MESSAGE request.

- Otherwise, the LLI checks whether data are ready to be sent (chained to the CCBTDATA chain).

  - If data are ready, a T-data.request is generated, and passed to the Transport layer.

  - If no data are ready, the CCB_ENABLE.TRANSM flag is set to true.

4.0 PROTOCOL STATE MACHINE

4.1 OVERVIEW

This module is in many ways the most important component of the Session layer. It carries the responsibility for the interpretation, validation and processing of events on all session connections.

The events basically come from two sources:

- Requests for service from local Session Users (SUs), and
- Messages transferred from other Session Services.

All of these are, at this stage, stored as Session Protocol Data Units (SPDUs), which are built according to the ISO protocol standard.

The interpretation consists on identifying the nature of each SPDU and any associated events (there may be more than one), and identifying the Session connection involved.

The validation consists on checking that the events may occur in the current state and with the current parameters (token disposition etc.) pertaining to the identified session connection.

The processing consists mainly on passing notifications of the events to the appropriate parties: either the local SU, or the remote session service, and on updating the state and parameter information stored in the session connection's CCB.

The precise nature of the processing involved differs in various ways, according to the nature of the event, the state of the connections, and the direction of flow of the SPDU.

The implementation of the Session protocol is, for each session connection, carried on by a non-deterministic finite state machine. A finite number of states, qualified by a number of parameters, are used to remember the condition of the connection. For each session connection, the state and the parameters are stored in the Connection Control Block (CCB).

The Session Protocol Machine implementation may be regarded as composed of three groups of procedures:

A. Those procedures concerned with interpretation and validation, which carry out pre-processing before the SPDU is passed on to:

B. the procedures which strctly implement the Session Protocol Machine (SPM), as defined by the standard, which carry out the protocol procedures and pass the result on to:

C. the procedures responsible for the disposition of the SPDU, which may be generated for the Session User, sent to the Transport layer, dropped or replaced by another SPDU.

When the SPDU is originated by a SU request, the final group of procedures must generate the final return code to be returned to the SU.

4.2 PROCEDURES OF GROUP (A)

These procedures are called either from the Session User, or from the Transport layer, and in both cases the involved Connection Identifier is specified as a parameter of the call. The Connection Identifier corresponds one-to-one with the CCB, so the state, parameters and actions to be performed on that connection are explicitly specified in the call.

The group A procedures have to analyze the SPDU message:

- recognizing the SPDU message category, enclosures, marks, information unit structure, and
- checking the validity of the SPDU message structure.

From the result of the SPDU message analysis, the SPM event is recognized.

4.3 PROCEDURES OF GROUP (B)

Any group (A) procedure, when calling a group (B) procedure, passes the following parameters:

- Connection Identifier (or CCB)
- SPDU (enqueued to the CCB)
- SPM event.

and receives the return code on completion of group (B) processing.

The group (B) processing consists on:

- matching the given event to the given state and parameters (extracted from the CCB), and

- according to the state-event combination, performing the state transition, and forwarding the SPDU to group (C) procedures.

Part of the group (B) procedures' code is described by the event-state decision tables which are given in "4.4 State Tables" pag. 118.

4.4 STATE TABLES

In "4.4.4 Formal description tables" pag. 135, the SPDU protocol interactions between two SPMs are described. That description defines states, events and actions which in this clause are consolidated into a formal description of the SPDU protocol by means of state-event tables.

4.4.1 Elements used in the formal description

Fig. 42 pag. 119 lists the states which are used in the formal description. For each entry there is a state code and a brief description.

Fig. 43 pag. 120, Fig. 44 pag. 121 and Fig. 45 pag. 122 list the events which are used in the formal description. For each entry there is a brief description.

Fig. 46 pag. 123 lists the SPM SPDU acronyms which are used in the formal description to identify SPDUs sent. In the formal description they are sometimes qualified by extra information in parenthesis.

The way in which these various elements are used is defined in "4.4.3 Description conventions" pag. 129.

The sequence for processing concatenated SPDUs is defined in "3.4.4 Concatenation" pag. 25 of "First Part."

Note: Fig. 42 pag. 119 and Fig. 43 pag. 120 include SPM states and SPM events which are not referenced in the text description. This reflects differencies in the level of detail between the text description and the formal description.

Code	Description
STA1	Idle no transport connection
STA1B	Wait for T_CONNECT.confirm
STA1C	Idle Transport connected
STA2A	Wait for ACCEPT SPDU
STA3	Wait for DISCONNECT SPDU
STA4A	Wait for MAJOR SYNC ACK SPDU
STA5A	Wait for RESYNCHRONIZE ACK SPDU
STA6	Wait for RESYNCHRONIZE ACK (collision) SPDU
STA7/13	Data transfer state
STA8	Wait for S-CONNECT.response event
STA9	Wait for S-RELEASE.response event
STA10A	Wait for S-SYNC-MAJOR.response event
STA11A	Wait for S-RESYNCHRONIZE.response event
STA15A	After PREPARE, wait for MAJOR SYNC ACK
STA15B	After PREPARE, wait for RESYNCHRONIZE
STA15C	After PREPARE, wait for RESYNCHRONIZE ACK
STA16	Wait for T-disconnect.indication event

Fig. 42. SPM states

Abbreviation	Description
AB-nr	ABORT (not reuse) SPDU
AC	ACCEPT SPDU
CN	CONNECT SPDU
DT	DATA TRANSFER SPDU
DN	DISCONNECT SPDU
TD	TYPED DATA SPDU
FN	FINISH SPDU
GT	GIVE TOKENS SPDU
MAA	MAJOR SYNC ACK SPDU
MAP	MAJOR SYNCHRONIZATION POINT SPDU
MIA	MINOR SYNC ACK SPDU
MIP	MINOR SYNCHRONIZATION POINT SPDU
NF	NOT FINISHED SPDU
PT	PLEASE TOKENS SPDU
PR-mcd	PREPARE (MAJ. SYNC ACK) SPDU
PR-ra	PREPARE (RESYNC ACK) SPDU
PR-rs	PREPARE (RESYNC) SPDU
RF-nr	REFUSE (not reuse) SPDU
RA	RESYNCHRONIZE ACK SPDU
RS-rst	RESYNCHRONIZE (restart) SPDU
RS-ab	RESYNCHRONIZE (abandon) SPDU
RS-set	RESYNCHRONIZE (set) SPDU
TCONind	T-CONNECT indication primitive
TCONconf	T-CONNECT confirmation primitive
TDISind	T-DISCONNECT indication primitive

Fig. 43. SPM incoming events (SPDUs and Transport indications)

Abbreviation	Description
SCONrec	S-CONNECT.request
SCONresp(+)	S-CONNECT(accept).request
SCONresp(-)	S-CONNECT(reject).request
SDTreq	S-DATA.request
STDreq	S-TYPED-DATA.request
SRELreq	S-RELEASE.request
SRELresp(+)	S-RELEASE(affirmative).request
SRELresp(-)	S-RELEASE(negative).request
SRSYNreq(ab)	S-RESYNCHRONIZE(abandon).request
SRSYNreq(rst)	S-RESYNCHRONIZE(restart).request
SRSYNreq(s)	S-RESYNCHRONIZE(set).request
SRSYNresp	S-RESYNCHRONIZE.response
SSYNMreq	S-SYNC-MAJOR.request
SSYNMresp	S-SYNC-MAJOR.response
SSYNmreq	S-SYNC-MINOR.request
SSYNmresp	S-SYNC-MINOR.response
SGTreq	S-TOKEN-GIVE.request
SPTreq	S-TOKEN-PLEASE.request
SUABreq	S-U-ABORT.request

Fig. 44. SPM incoming events (from the SS user)

Abbreviation	Description
AB-nr	ABORT (not reuse) SPDU
AC	ACCEPT SPDU
CN	CONNECT SPDU
DT	DATA TRANSFER SPDU
DN	DISCONNECT SPDU
TD	TYPED DATA SPDU
FN	FINISH SPDU
GT	GIVE TOKENS SPDU
MAA	MAJOR SYNC ACK SPDU
MAP	MAJOR SYNCHRONIZATION POINT SPDU
MIA	MINOR SYNC ACK SPDU
MIP	MINOR SYNCHRONIZATION POINT SPDU
NF	NOT FINISHED SPDU
PT	PLEASE TOKENS SPDU
PR-mcd	PREPARE (MAJ. SYNC ACK) SPDU
PR-ra	PREPARE (RESYNC ACK) SPDU
PR-rs	PREPARE (RESYNC) SPDU
RF-nr	REFUSE (not reuse) SPDU
RA	RESYNCHRONIZE ACK SPDU
RS-rst	RESYNCHRONIZE (restart) SPDU
RS-ab	RESYNCHRONIZE (abandon) SPDU
RS-set	RESYNCHRONIZE (set) SPDU
TCONind	T-CONNECT indication primitive
TCONconf	T-CONNECT confirmation primitive
TDISind	T-DISCONNECT indication primitive

Fig. 43. SPM incoming events (SPDUs and Transport indications)

Abbreviation	Description
SCONrec	S-CONNECT.request
SCONresp(+)	S-CONNECT(accept).request
SCONresp(-)	S-CONNECT(reject).request
SDTreq	S-DATA.request
STDreq	S-TYPED-DATA.request
SRELreq	S-RELEASE.request
SRELresp(+)	S-RELEASE(affirmative).request
SRELresp(-)	S-RELEASE(negative).request
SRSYNreq(ab)	S-RESYNCHRONIZE(abandon).request
SRSYNreq(rst)	S-RESYNCHRONIZE(restart).request
SRSYNreq(s)	S-RESYNCHRONIZE(set).request
SRSYNresp	S-RESYNCHRONIZE.response
SSYNMreq	S-SYNC-MAJOR.request
SSYNMresp	S-SYNC-MAJOR.response
SSYNmreq	S-SYNC-MINOR.request
SSYNmresp	S-SYNC-MINOR.response
SGTreq	S-TOKEN-GIVE.request
SPTreq	S-TOKEN-PLEASE.request
SUABreq	S-U-ABORT.request

Fig. 44. SPM incoming events (from the SS user)

Code	Description
SCONind	S-CONNECT.indication
SCONconf(+)	S-CONNECT(accept).confirmation
SCONconf(-)	S-CONNECT(reject).confirmation
SDTind	S-DATA.indication
STDind	S-TYPED-DATA.indication
SRELind	S-RELEASE.indication
SRELconf(+)	S-RELEASE(affirmative).indication
SRELconf(-)	S-RELEASE(negative).indication
SRSYNind	S-RESYNCHRONIZE.indication
SRSYNconf	S-RESYNCHRONIZE.confirmation
SSYNMind	S-SYNC-MAJOR.indication
SSYNMconf	S-SYNC-MAJOR.confirmation
SSYNmind	S-SYNC-MINOR.indication
SSYNmconf	S-SYNC-MINOR.confirmation
SGTind	S-TOKEN-GIVE.indication
SPTind	S-TOKEN-PLEASE.indication
SABind	S-P-ABORT or S-U-ABORT indication
TCONreq	T-CONNECT request primitive
TCONresp	T-CONNECT response primitive
TDISreq	T-DISCONNECT request primitive

Fig. 45. SPM generated events

SPDU name	SPDU code
CONNECT	CN
ACCEPT	AC
REFUSE	RF
FINISH	FN
DISCONNECT	DN
ABORT	AB
ABORT ACCEPT	AA
DATA TRANSFER	DT
PLEASE TOKENS	PT
GIVE TOKENS	GT
TYPED DATA	ED
NOT FINISHED	NF
MINOR SYNCHRONIZATION POINT	MIP
MINOR SYNC ACK	MIA
MAJOR SYNCHRONIZATION POINT	MAP
MAJOR SYNC ACK	MAA
PREPARE	PR
RESYNCHRONIZE	RS
RESYNCHRONIZE ACK	RA

Fig. 46. Protocol messages acronyms

4.4.2 Definitions

Functional Units

The set of all existing Functional Units is defined as:

fu-dom = {FD, HD, EXCEP, TD, NR, SY, MA, RESYN, EX, ACT, CD}

Note: This set lists all the Functional Units defined in the ISO Session Protocol, although not all of them are implemented in OSIRIDE version 1. A boolean function is defined as follows:

For any f in **fu-dom**:

FU(f) = true if and only if the Functional Unit f has been selected at connection time.

Tokens

The set of tokens defined in this protocol is defined as:

tk-dom = {mi, ma, tr, dk}

Where:

mi Minor synchronize Token

ma Major/activity token

tr Release token

dk Data token

The following boolean functions are defined on the tk-dom:

- AV(t), which defines the availability of the corresponding token, and has the following values:

  AV(mi) := FU(SY)

  AV(dk) := FU(HD)

  AV(tr) := FU(NR)

  AV(ma) := FU(MA) or FU(ACT)

- OWNED(t), which defines the assignment of the corresponding token, and is defined as:

  OWNED(t) := true if token assigned to the SPM

  OWNED(t) := false if token not assigned to the SPM

  OWNED(t) is not defined if AV(t) = false.

  OWNED(t) is set when:

  1. the ACCEPT SPDU is sent or received; or
  2. the RESYNCHRONIZE ACK SPDU is sent or received; or
  3. the GIVE TOKENS SPDU is sent or received

- I(t) := ¬AV(t) or OWNED(t)
  This function, when true, indicates that the SPM has initiating rights for the behaviour controlled by the token. This applies even if the corresponding token is not available.

- A(t) := ¬AV(t) or ¬ OWNED(t)
  This function, when true, indicates that the SPM has accepting rigths for the behaviour controlled by the token. This applies even if the corresponding token is not available.

- II(t) := AV(t) and OWNED(t)
  This function, when true, indicates that the SPM has initiating rigths as I(t), but this applies to the case when the

behaviour may only be initiated if the corresponding is available and owned.

- **AA(t) := AV(t) and ¬OWNED(t)**
  This functions, when true, indicates that the SPM has accepting rigths as A(t), but only if the corresponding token is available, but not owned.

Set of tokens

A Set of tokens is a subset of tk-dom. It can be defined by the list of its elements or by the name of a predefined set as for example:

RT = {tokens received in a SPDU and, by implication, indicated as received}

GT = {tokens sent in a SPDU and, by implication, named in a request}

Two boolean functions are defined:

- **ALL(f,Set)**
  With f any of {AV, OWNED, I, A, II, AA} is:
  ALL(f,Set) := AND (f(t))

  Example:

  ALL (A,tk-dom) = **none of the defined tokens are owned**

- **ANY(f,Set)**

  ANY(f,Set) := OR (f(t))

  Example:

  ANY(I,tk-dom) = **at least one of the available tokens is owned.**

Variables

TEXP This function is set **true** when the Iransport expedited service is selected for use on this Session connection. This is **always** the case in OSIRIDE Version 1.

VACT This is a boolean variable having the following values:

- **true** if the Activity management Functional Unit has been selected (FU(ACT) := true) and an activity is in progress.
- **false** if the Activity management Functional Unit has been selected (FU(ACT) := true) and no activity is in progress.
- the value is not defined if FU(ACT) := false.

Note: In OSIRIDE version 1, the value is **never** defined. VACT is set as follows:

1. At connection establishment time, if the Activity management Functional Unit has been selected, VACT is set to **false**. Otherwise, VACT is not set. **This is the OSIRIDE case.**

2. VACT is set to **true** when the ACTIVITY START SPDU is sent or received (FU(ACT) = true). **In OSIRIDE version 1, this will never happen.**

3. VACT is set to **false** when an activity is abnormally terminated (completion of ACTIVITY DISCARD or ACTIVITY INTERRUPT). **In OSIRIDE version 1, this will never happen.**

4. VACT is set to **Vnextact** (see below), when a MAJOR SYNC ACK SPDU or an ACTIVITY END ACK SPDU is sent or received.

Vnextact

This is a boolean variable which is used when the Activity management Functional Unit is selected (FU(ACT) = true). **This is not the case in OSIRIDE Version 1.**

It is used to indicate the next value of VACT when a MAJOR SYNC ACK SPDU or an ACTIVITY END ACK SPDU is sent or received.

Vnextact is set when a MAJOR SYNC POINT SPDU or an ACTIVITY END SPDU is sent or received:

1. If FU(ACT) = true, and an ACTIVITY END SPDU is sent or received, then Vnextact is set to **false**. **In OSIRIDE version 1, this will never happen.**

2. If FU(ACT) = true, and a MAJOR SYNC POINT SPDU is sent or received, then Vnextact is set to **true**. **In OSIRIDE version 1, this will never happen.**

Vrsp and Vrspnb

These variables are used to solve the resynchronization collisions.

Vrsp indicates what kind of resynchronization is in progress:

Vrsp = no no resynchronization in progress

Vrsp = r restart

Vrsp = a abandon

Vrsp = s set

Vrsp = int interrupt activity

Vrsp = dsc discard activity

Vrspnb indicates the serial number in case of resynchronize restart.

Vrsp and, if necessary, Vrspnb are set when a RESYNCHRONIZE SPDU, ACTIVITY INTERRUPT SPDU, or an ACTIVITY DISCARD SPDU is sent or received. Vrsp is set to **no** when the SPM goes to STA7/13.

SPMwinner

SPMwinner is a boolean function which is used in case of resynchronization collision, i.e.:

1. a RESYNCHRONIZE SPDU is received and Vrsp is not equal to **no**.
2. an S-RESYNCHRONIZE.request is received and Vrsp is not equal to **no**.

The SPMwinner condition is true if the SPM (which holds the current resynchronization) wins against the new colliding event.

The SPMwinner condition is calculated as follows:

1. The next Vrsp and Vrspnb values are evaluated according to the parameters of the received event. The new calculated value for Vrsp is compared to the current Vrsp with the following ordering rules:

   dsc prevails over **int**

   int prevails over **a**

   a prevails over **s**

   s prevails over **r**

   If both are equal to **r**, then the new calculated value for Vrspnb is compared to the current Vrspnb and the lower value prevails.

2. If the current value of Vrsp (and Vrspnb if necessary) prevails, then the SPMwinner condition is true (in this case, the current resynchronization prevails over the colliding one).

3. If the current value of Vrsp (and Vrspnb if necessary) does not prevail, then the SPM-winner condition is **false** (in this case, the colliding resynchronization prevails over the current one).

4. If the above comparisons result in equality, and if the colliding event has been generated by the initiator of the Session connection (either a RESYNCHRONIZE SPDU was received from the session connection initiator or a local S-RESYNCHRONIZE.request was issued by the session connection initiator), the SPMwinner condition is **false**.

If the SPM is winner (SPMwinner condition is **true**), then the current resynchronization wins against the colliding one, and Vrsp and Vrspnb are not updated.

If the SPM is not winner (SPMwinner condition is **false**), then the colliding resynchronization is taken into account, and Vrsp and Vrspnb are updated.

Vtca

This boolean variable has the following values:

- **false** if the SPM initiated the T-CONNECT.request (transport connection initiator).
- **true** if the SPM received the T-CONNECT.indication (transport connection acceptor).

Vtrr

This boolean variable has the following values:

- **true** if the transport connection can be reused by the SPM for another session connection. **This will never happen in OSIRIDE version 1.**
- **false** if the transport connection cannot be reused by the SPM for another session connection. **This is always the case in OSIRIDE version 1.**

Vsc

This boolean variable is used to control the rigth to issue synchronization point confirmations.

The variable Vsc has the following values:

- **true** if the session user has the right to issue synchronization point confirmations when V(A) is less than V(M);
- **false** if the session user does not have the right to issue synchronization point confirmations.

Vsc is set **false** when a MINOR SYNC POINT SPDU is sent. Vsc is set **true** when a MINOR SYNC POINT SPDU is received.

Vcoll

This boolean variable has the following values:

- **true** if a collision of FINISH SPDUs has been detected.
- **false** if no collision of FINISH SPDUs has been detected.

The variable is set to **false** during the Session connection establishment phase.

4.4.3 Description conventions

The formal description is in the tables from Fig. 54 to Fig. 70.

The horizontal dimension of each table is the set of all the states. For each state there is an entry (i.e. a column). If, for any given state, there is no valid event, then the state does not appear in the table (i.e. no column).

A state which does not appear in a given table may appear in another table which has different applicability. Likewise for an intersection, which is invalid in a given table, may be valid in another table which has different applicability.

The vertical dimension of each table is the set of all relevant SPM request events, incoming SPDU events and SPM response events. For each event there is an entry (i.e. a row).

Each valid intersection contains:

A. one or more predicates (where relevant);

B. one or more actions (where relevant);

C. the new state (always).

The applicable predicates are listed immediately after each table.

The action generally consists of sending a SPDU or issuing a local primitive event (indication event or confirmation event). Sometimes the event causing an action is locally enqueued, or causes other events to be dequeued or cancelled. The actions are listed in the tables of Fig. 47 pag. 130 and Fig. 48 pag. 131.

The new state is the state which is entered after the specified action is completed. All dequeued events are considered by the SPM with priority over any other incoming events.

An invalid intersection is shown by a blank entry. Two different actions are taken when entering an invalid intersection:

1. If the event which caused the error is an SS user generated event, an error condition is returned. No other action is taken.

2. If the event which caused the error is due to an SPDU coming form the remote SPM:

   A. An ABORT SPDU is sent, with reason code = Protocol Error.

   B. A P_ABORT.indication is generated for the SS user, with reason code= Protocol Error.

   C. The Abort Timer, TAB, is started, and the new state is STA16.

Code	Action
[1]	Vtca := true
[2]	Vtca := false
[3]	stop timer TAB
[4]	start timer TAB
[5]	Set V(A):= V(M):= serial number in ACCEPT SPDU Set V(R):= 0 Set Vcoll:= false Set Vrsp:= no
[6]	Recall the events until the queue has been completely examined
[7]	Set Vtrr := true
[8]	Set Vtrr := false
[9]	Set Vtrr according to the Transport Disconnect PV field in the SPDU. As a local decision, Vtrr is **always** set false.
[10]	Store the event in the queue.
[11]	Update the position of the tokens.
[12]	Set VACT := true
[13]	Set Vnextact.
[14]	Set VACT:= Vnextact
[15]	Clear the queue.
[16]	Update Vrsp and, if RS-r, Vrspnt

Fig. 47. Actions taken by the SPM

Code	ction
[17]	Not used
[18]	Set Vcoll:= **true**
[19]	V(M) := maximum(V(M), received serial number)
[20]	Set Vsc:= **false**
[21]	Set V(M):= V(M) + 1
[22]	Set V(R):= V(A) := V(M)
[23]	Set V(M) := V(M) + 1 If Vsc:= **false**, then Set V(A) := V(M) Set Vsc := true
[24]	Set V(M) := V(M) + 1 If Vsc:= **true**, then Set V(A) := V(M) Set Vsc := false
[25]	Set V(A) = Serial number + 1
[26]	Set V(A) := V(R) := V(R) := 1
[27]	Set V(A) := V(M) := Serial number +1 Set V(R) :=1
[28]	Set V(M) := V(A) := Serial number If Vrsp := **a** then Set V(R) := 0 If Vrsp := **s** then Set V(R) := 0 Set Vrsp := **no**
[29]	Set the position of the tokens sc that all available tokens are owned and set VACT := **false** Set Vrsp := **no.**
[30]	Set the position of the tokens so that all available tokens are not owned and set VACT := **false** Set Vrsp := **no.**
[31]	Set V(M):= V(M) + 1 If Vsc:= **false** then set V(A) := V(M)

Fig. 48. Actions taken by the SPM (continued)

Code	Action
P01	¬Vtca
P02	local choice & ¬TEXP
P03	I(dk)
P04	FU(FD) & ¬Vcoll
P05	A(dk)
P06	FU(TD)
P07	FU(TD) & ¬Vcoll
P08	FU(EX)
P09	FU(EX) & ¬Vcoll
P10	¬Vcoll
P11	MAP not AE
P12	(¬FU(ACT) OR Vact) & A(dk) & A(mi) &AA(ma)
P13	(¬FU(ACT) OR Vact) & I(dk) & I(mi) &II(ma)
P14	(¬FU(ACT) OR Vact) & A(dk) & AA(mi)
P15	(¬FU(ACT) OR Vact) & I(dk) & II(mi)
P16	¬TEXP
P17	(¬FU(ACT) OR Vact) & FU(SY) & ¬Vsc
P18	(¬FU(ACT) OR Vact) & FU(SY) & Vsc

Fig. 49. Predicates

Note: P33 and P56 are unused

Code	Action
P19	Serial number = V(M)
P20	Serial number = V(M)-1
P21	V(M) > serial number >= V(A)
P22	Vsc
P23	FU(ACT) & ¬Vnextact
P24	¬SPMwinner
P25	(FU(SY) OR FU(MA)) & FU(RESYN)
P26	(¬FU(ACT) OR Vact)
P27	Vrsp = no
P28	FU(RESYN)
P29	(¬FU(ACT) OR Vact) & FU(RESYN)
P30	¬FU(ACT) OR Vnextact
P31	FU(ACT) & Vnextact
P32	serial number >= V(R)
P34	FU(ACT)
P35	FU(RESYN) & ¬TEXP
P36	FU(RESYN) & TEXP

Fig. 50. Predicates (continued).

Code	Action
P37	FU(ACT) & TEXP
P38	FU(ACT) & ¬TEXP
P39	Vact & II(ma)
P40	AA(ma)
P41	Vrsp = dsc
P42	Vrsp = int
P43	((Vrsp = r) & serial number = Vrspnb)
P44	(FU(ACT) OR ¬Vact) & A(dk) & A(mi) & A(ma)
P45	(FU(ACT) & ¬Vact) & I(dk) & I(mi) & I(ma)
P46	FU(CD) & (FU(ACT) & ¬Vact) & A(dk) & A(mi) & ¬OWNED(ma)
P47	FU(CD) & (FU(ACT) & ¬Vact) & I(dk) & I(mi) & OWNED(ma)
P48	FU(EXCEP) & FU(HD)
P49	(¬FU(ACT) OR Vact)
P50	FU(EXCEP) & (¬FU(ACT) OR Vact) & AA(dk)
P51	FU(EXCEP) & (¬FU(ACT) OR Vact) & II(dk)
P52	FU(EXCEP) & ¬FU(ACT) & II(dk)
P53	ALL(AV,RT)
P54	ALL(II,GT)
P55	(FU(ACT) & ¬Vact) & ANY(II,tk-dom)

Fig. 51. Predicates (2nd continued)

Code	Action
P57	ALL(II,GT) & (dk not in GT)
P58	ALL(II,GT) & (dk in GT)
P59	ALL(AA,GT)
P60	ALL(AA,GT) & (dk not in GT)
P61	ALL(AA,GT) & (dk in GT)
P62	(FU(ACT) & ¬Vact) & ANY(AA,tk-dom)
P63	ALL(I,tk-dom) & (¬FU(ACT) OR ¬Vact)
P64	local choice & ¬Vtca & ¬TEXP
P65	ANY(AV,tk-dom) & (¬FU(ACT) OR ¬Vact)
P66	Vtrr
P67	FU(NR)
P68	ALL(A,tk-dom) & (¬FU(ACT) OR ¬Vact)
P69	Vcoll
P70	FU(FD)
P71	FU(ACT) & Vact & I(dk) & I(mi) & II(ma)
P72	FU(ACT) & Vact & A(dk) & A(mi) & AA(ma)

Fig. 52. Predicates (3rd continued)

4.4.4 Formal description tables

The session protocol is described by using a number of tables; this introduces redundancy but allows individual descriptions of each sub-protocol. For editing reasons, in some cases, the same table cannot contain all the states: then continuation tables are used.

In order to know how a SPM should behave when it is in a given state (STA i) and when a given event (EVE j) occurs, find the table(s) with the appropriate applicability by using the index in

Fig. 53 pag. 136. If the appropriate table contains the state (STA i) and the event (EVE j), the action and the new state etc. are defined in the intersection ((STA i),(EVE j)). Otherwise, the combination ((STA i),(EVE j)) is invalid.

Table	Sub-protocol
Fig. 54	Connection
Fig. 55	Data transfer
Fig. 56	Data transfer
Fig. 57	Minor Synchronization
Fig. 58	Minor Synchronization
Fig. 59	Major Synchronization
Fig. 60	Major Synchronization
Fig. 61	Resynchronize
Fig. 62	Resynchronize
Fig. 63	Resynchronize
Fig. 64	Resynchronize
Fig. 65	Disconnection
Fig. 66	Disconnection
Fig. 67	Abort
Fig. 68	Abort
Fig. 69	Token Transfer
Fig. 70	Token Transfer

Fig. 53. Index of formal description tables

	STA1	STA1B	STA1C	STA2A	STA8	STA16
SCONreq	TCONreq [2] STA1B		P01: send CN STA2A			
SCONresp(+)					send AC [5] STA7/13	
SCONresp(-)					¬P02: send RF-n [4] STA16 P02: send RF-r STA16	
CN			¬P01: SCONind STA8 P1: TDISreq STA1			TDISreq [3] STA1
AC			TDISreq STA1	SCONcnf+ [5] STA7/13 [6]		STA16
RF-nr			TDISreq STA1	SCONcnf- TDISreq STA1		STA16
TCONconf		send CN STA2A				
TCONind	TCONresp [1] STA1C					

Fig. 54. Connection protocol

	STA1C	STA3	STA4A	STA5A	STA6	STA7/13
SDTreq						P03: send DT STA7/13
DT	TDISreq STA1	P5&P10: SDTind STA3	P70: SDTind STA4A	P5 STA5A	P5 STA6	P5: SDTind STA7/13
TD	TDISreq STA1	P6&P10: STDind STA3	P6: STDind STA4A	P6: STA5A	P6 STA6	P6: STDind STA7/13
STDreq						P6: send TD STA7/13

Fig. 55. Data Transfer

	STA9	STA10A	STA15A	STA15B	STA16
SDTreq	P4: send DT STA9	P3: send DT STA10A		 STA15B	
DT			P70: SDTind STA15A	P5: STA15B	 STA16
STDreq	P7: send TD STA9	P6: send TD STA10A		P6: STA15B	
TD			P6: STDind STA15A	P6: STA15B	P6 STA16

Fig. 56. Data Transfer (continued)

	STA3	STA4A	STA5A	STA6	STA7/13	STA9	STA10A
SSYNmreq					P15: send MIP [24] STA7/13		
SSYNmrsp					P18]P21 send MIA [25] STA7/13	P18&P21 send MIA [25] STA9	P18&P21 send MIA [25] STA10A
MIP			P14 STA5A	P14 STA6	P14&P19 SSYNmind [23] STA7/13		
MIA	P17&P21 SSYNmcnf [25] STA3	P17&P21 SSYNmcnf [25] STA4A	P17 STA5A	P17 STA6	P17&P21 SSYNmcnf [25] STA7/13		

Fig. 57. Minor synchronization

	STA15A	STA15B	STA15C	STA16
SSYNmreq		P15 STA15B		
SSYNmrsp		P18&P21 STA15B		
MIP		P14 STA15B	P14 STA15C	STA16
MIA	P17&P21 SSYNmcnf [25] STA15A	P17 STA15B	P17 STA15C	 STA16

Fig. 58. Minor synchronization (continued)

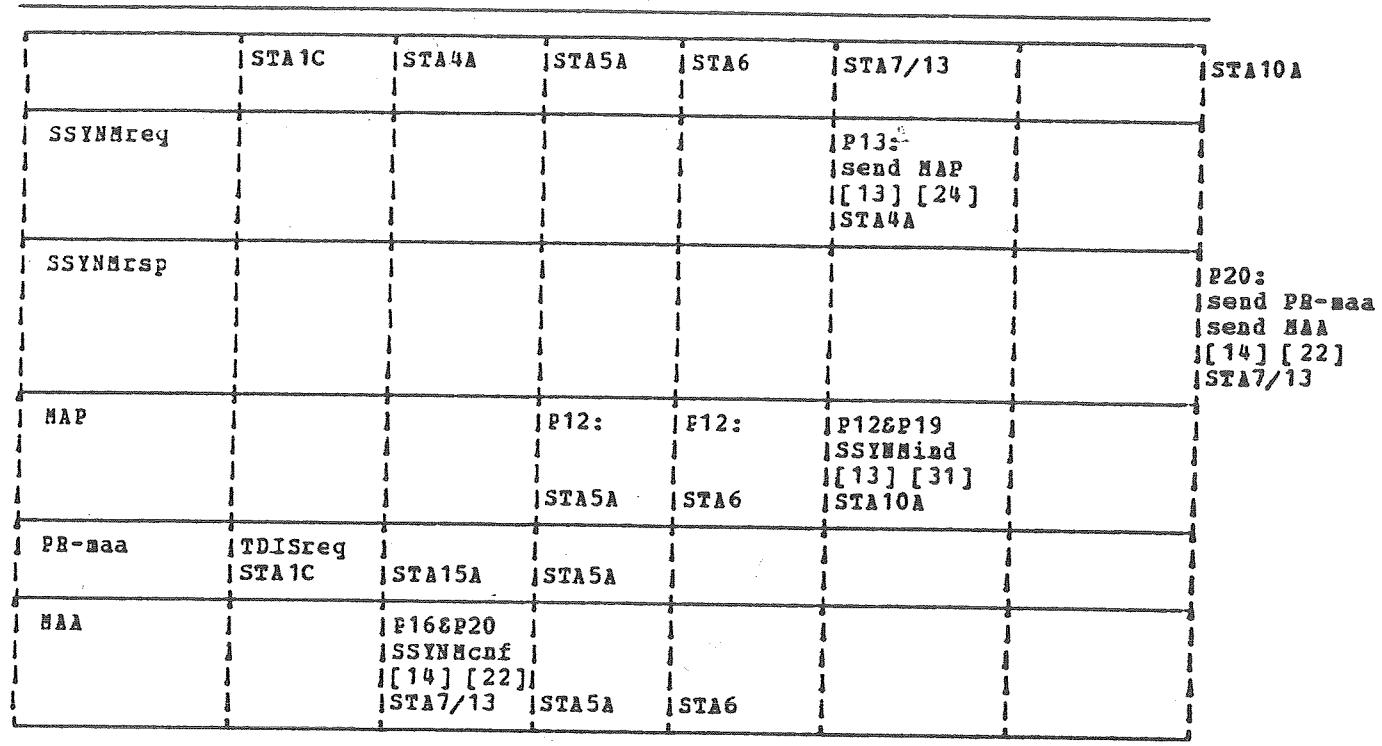


Fig. 59. Major synchronization

	STA15A	STA15B	STA15C	STA16
SSYNMreq		P13: STA15B		
SSYNMrsp		P20: STA15B		
MAP		P12: STA15B	P12: STA15C	STA16
PR-maa				STA16
MAA	P20&¬P23 SSYNMcnf [14][22] STA7/13[6]	STA15B	STA15C	STA16

Fig. 60. Major synchronization (continued)

	STA4A	STA7/13	STA9
SRSYNreq(r)		P25&P26&P32 send PR-rs send RS-r [16] STA5A	P25&¬P34&P32 send PR-rs send RS-r [16] STA5A
SRSYNreq(a)	P28: send PR-rs send RS-a [16] STA5A	P29 send PR-rs send RS-a [16] STA5A	P28&¬P34 send PR-rs send RS-a [16] STA5A
SRSYNreq(s)	P28: send PR-rs send RS-s [16] STA5A	P25&¬P34 send PR-rs send RS-s [16] STA5A	P25&P26: send PR-rs send RS-s [16] STA5A
SRSYNrsp			

Fig. 61. Resynchronization

	STA2A	STA3	STA4A	STA5A	STA6	STA7/13
PR-rs	[10] STA2A	STA15B	STA15B	STA6	[10] STA6	STA15B
PR-ra				STA15C	[10] STA6	
ES-r		¬P34&P35&P32 SRSYNind [16] STA11A	P32&P35 SRSYNind [16] STA11A	¬P24&P32&P35 STA5A P24&P32&P35 SRSYNind [16] STA11A	¬P24&P32 STA5A[6] P24&P32 SRSYNind [16] STA11A[6]	P32&P26&P35 SRSYNind [16] STA11A
ES-a		¬P34&P35 [19] SRSYNind [16] STA11A	&P35 [19] SRSYNind [16] STA11A	¬P24&P35 STA5A P24&P35 [19]SRSYNind [16] STA11A	¬P24 STA5A[6] P24 SRSYNind [16] STA11A[6]	P26&P35 [19] SRSYNind [16] STA11A
ES-s		¬P34&P35 SRSYNind [16] STA11A	P35 SRSYNind [16] STA11A	¬P24&P35 STA5A P24&P35 [19]SRSYNind [16] STA11A	¬P24 STA5A[6] P24 SRSYNind [16] STA11A[6]	P26&P35 SRSYNind [16] STA11A
RA				P35&P43 SRSYNcnf [28][11] STA7/13		

Fig. 62. Resynchronization (continued)

	STA10A	STA11A	STA15A	STA15B
SRSYNreq(r)	P25&P32 send PR-rs send RS-r [16] STA5A	P24&P32 send PR-rs send RS-r [16] STA5A		P25&P27&P32 send PR-rs send RS-r [16] STA6
SRSYNreq(a)	P28 send PR-rs send RS-a [16] STA5A	P24 send PR-rs send RS-a [16] STA5A	P28&P30 send PR-rs send RS-a [16] STA5A	P27&P28 send PR-rs send RS-a [16] STA6
SRSYNreq(s)	P25 send PR-rs send RS-s [16] STA5A	P24 send PR-rs send RS-s [16] STA5A	P28&P30 send PR-rs send RS-s [16] STA5A	P25&P27 send PR-rs send RS-s [16] STA6
SRSYNrsp		P43 send PR-ra send RA [28][11] STA7/13		

Fig. 63. Resynchronization (2nd continued)

	STA10A	STA15A	STA15B	STA15C	STA16
PR-rs	STA15B	[10] STA15A		[15] STA15B	
PR-ra					STA16
RS-a	P35 [19] SRSYNind [16] STA11A		P29 [19] SRSYNind [16] STA11A		STA16
RS-r			P32&P29 [19] SRSYNind [16] STA11A		STA16
RS-s	P35 SRSYNind [16] STA11A		P29 SRSYNind [16] STA11A		STA16
RA			STA15B	P36&P43 SRSYNconf [28] [11] STA7/13[6]	STA16

Fig. 64. Resynchronization (3rd continued)

	STA1C	STA3	STA5A	STA6	STA7/13
SRELreq					P63&¬P64 FN-nr[8] STA3 P63&P64 FN-r[7] STA3
SRELrsp+					
SRELresp-					
FN-nr	TDISreq STA1	¬P65 SRELind [8][18] STA9	P68 STA5A	P68 STA6	P68 SRELind [8] STA9
DN	TDISreq STA1	¬P66 SRELcnf+ TDISreq STA1 P66 SRELcnf+ STA1C			
NF	TDISreq STA1	P67 SRELcnf- STA7/13			

Fig. 65. Disconnect

	STA9	STA15B	STA15C	STA16
SRELreq	¬P65 FN-nr [8][18] STA9	P63 STA15B		
SRELrsp+	¬P66 send DN[4] STA16 P66&¬P69 send DN STA1C P66&P69 send DN[8] STA3			
SRELrsp-	P67 send NF STA7/13			
FN-nr			P68 STA15C	STA16
DN	P69 SRELcnf+ STA9			 STA16
NF		P67 SRELcnf- STA15B		 STA16

Fig. 66. Disconnect (continued)

	STA1B	STA1C	STA2A	STA3	STA4A	STA5A	STA6	STA7/13
AA		TDISreq STA1						
AB-nr		TDISreq STA1	SABind TDISreq STA1	SABind TDISreq STA1	SABind TDISreq STA1	SABind TDISreq STA1	SABind TDISreq STA1	SABind TDISreq STA1
SUABreq	TDISreq STA1		¬P2 AB-nr [4] STA16	¬P2 AB-nr [4] STA16	¬P2 AB-nr [4] STA16	¬P2 AB-nr [4] STA16	¬P2 AB-nr [4] STA16	¬P2 AB-nr [4] STA16
TDISind	SPABind STA1	 STA1	SPABind STA1	SPABind STA1	SPABind STA1	SPABind STA1	SPABind STA1	SPABind STA1

Fig. 67. Abort and Transport Disconnection

	STA8	STA9	STA10A	STA11A	STA15A	STA15B	STA15C	STA16
AA								[3] TDISreq STA1
AB-nr	SABind TDISreq STA1	SABind TDISreq STA1	SABind TDISreq STA1	SABind TDISreq STA1	SABind TDISreq STA1	SABind TDISreq STA1	SABind TDISreq STA1	[3] TDISreq STA1
SABreq	¬P2 AB-nr [4] STA16	¬P2 AB-nr [4] STA16	¬P2 AB-nr [4] STA16	¬P2 AB-nr [4] STA16	¬P2 AB-nr [4] STA16	¬P2 AB-nr [4] STA16	¬P2 AB-nr [4] STA16	
TDISind	SPABind STA1	SPAEind STA1	SPABind STA1	SPABind STA1	SPABind STA1	SPABind STA1	SPABind STA1	[3] STA1

Fig. 68. Abort and Transport Disconnection (continued)

	STA1C	STA3	STA4A	STA5A	STA6	STA7/13	STA9
SGTreq			P54 send GT [11] STA4A			P54 send GT [11] STA7/13	
SPTreq						P53 send PT STA7/13	P53 send PT STA9
PT	TDISreq STA1	P53 SPTind STA3	P53 SPTind STA4A	P53 STA5A	P53 STA6	P53 SPTind STA7/13	
GT	TDISreq STA1		P59 SGTind [11] STA4A	P59 STA5A	P59 STA6	P59 SGTind [11] STA7/13	

Fig. 69. Token transfer

	STA10A	STA15A	STA15B	STA15C	STA16
SPTreq	P53 send PT STA10A		P53 STA15B		
SGTreq	P54 send GT [11] STA10A				
PT		P53 SPTind STA15A	P53 STA15B	P53 STA15C	STA16
GT		P59 SGTind [11] STA15A	P59 STA15B	P59 STA15C	 STA16

Fig. 70. Token transfer (continued)

4.5 PROTOCOL ROUTINES

For every event-state combination, a state transition is defined, which is generally processed as an atomic activity. Only a few state transitions are represented by a sequence of asynchronous activities.

The translation from the state tables description into high level language processable activities is straightforward, but the entire description of the whole set of activities is rather voluminous. So, for the sake of a good visibility of the overall "session what-to-do" and "session how-to-do", only a few very instructive examples have been choosen ("4.5.1 Connection Establishment (SU request)" pag. 149, "4.5.3 Connection establishment (Protocol message)" pag. 150 and "4.5.5 Session connection abortion" pag. 152). All other cases can be reduced to these complex cases.

4.5.1 Connection Establishment (SU request)

When a request is received from a SU to establish a new session connection, then the Connection Handler (CH) task is involved. The Connection Handler converts the user-supplied address, which is in the form **<hostid> <application id>**, into the form used for transport connection establishment.

A new CCB has already been allocated by the ULI. At this point, the implementation strategy may differ according to the different operating system: as, in fact, the translation is generally performed by means of a table which is read from disk, this activity of connection establishment might be suspended here, awaiting for translation. If, however, another implementation strategy is choosen, there is no need to suspend this task here.

In any case, the most general case is the asynchronous one, so that what follows will describe an asynchronous way of operation.

At a certain time, when a reply is received from the CH, the connection establishment procedure, is resumed.

If the CH response is **failure**, the SU is informed that the provided address is invalid, by means of a **MESSAGE**, the SPDU is dropped, and the CCB is de-allocated.

If the CH response is **success**, the CH returns the transport layer address of the required host, which is placed in the CCBSYSTEM field of the CCB. Then, a group (B) procedure is called, with the following parameters:

CCB

SPM event

This is another asynchronous process, because it embraces a Transport Connection establishment.

In the case when a Transport Connection cannot be established, the SPDU is dropped, the CCB is de-allocated, and the SU is informed by means of a **MESSAGE.**

If the Transport connection was successfully established, then the required state transition is carried on by the corresponding group (B) procedure, which causes the SPDU to be forwarded to the Transport layer (group (C) procedure). The S_CONNECT.request is terminated successfully (this does not yet mean that the session connection has been successfully established).

At this point, from the point of view of the local session service, the connection is **active**, and all subsequent event are processed as described in "4.5.2 SU requests - All other" pag. 150.

4.5.2 SU requests - All other

Once a CCB has been allocated, all other SU requests are processed as a single activity, differently from the case of connection establishment.

The procedure is always the following:

- If the event is valid, given the current CCB state and parameters (validation performed by a group (A) procedure), then:
  - The state transition described in "4.4 State Tables" pag. 118 is performed, and
  - the SPDU is forwarded on to the Transport layer.

The SU request may be rejected by the SPM either because of some errors (e.g. request invalid in the current state), or because a high priority SPDU is awaiting to be delivered on that session connection. In all these cases, the SPDU is simply discarded, and an appropriate return code is sent back to the SU by means of a **MESSAGE.**

When the SU request is accepted, the success return code is reported in the same way.

4.5.3 Connection establishment (Protocol message)

The SPM task receives notification of a remotely initiated Session connection establishment through the CONNECT session protocol message, received as Transport Data from the Transport Layer.

A preliminary allocation of a "dummy" CCE is needed by the Session layer, in order to handle incoming requests.

When, in this example, the group (B) procedure is invoked, the CONNECT SPDU is chained to this dummy CCB.

One of the items contained in the CONNECT SPDU is the "Acceptor Address" item (see also "5.0 Structure and encoding of SPDUs" pag. 51 in "First Part"), a string up to 16 characters. The matching of this string with the SU supplied **name** in the OPEN primitive (see "3.1 Upper Layer Interface" pag. 85) is carried out asynchronously by the Connection Handler task. The same considerations on "asynchronicity" expressed in "4.5.1 Connection Establishment (SU request)" pag. 149 are valid here.

The CH is requested to perform the address translation. When a reply is received, it will be either:

- Success:
  - The address of a UCB is returned, corresponding to the requested user.
  - The CCB is removed from the dummy CCE chain, and chained to the user UCE.
  - The SPDU is then processed by the group (B) procedure (EVE3 and STA1).
  - The connection indication (prepared by a group (C) procedure), is finally passed on to the SU by means of a **MESSAGE** function.
- OPEN not yet issued (SU not found):
  - Some special, implementation dependent, actions may be taken in this case, either to "load and start" the requested user task, or to suspend operations for that request until an OPEN is not issued.

    The "normal" CSIRIDE behaviour, however, is just to consider this case as a failure, as the next case in this list, specifying the ISO return code "user not attached".
- Failure:
  - A REFUSE protocol message is built and sent on the Transport connection, carrying on an appropriate return code. This incidentally leads to Transport disconnection.

If the whole process has been without errors, then from the local point of view the session connection has been established, since the CCB has been allocated, chained to the UCB, and initialized.

All further incoming messages are dealt with as in the next section.

4.5.4 Session protocol messages - All others

All session protocol messages on active Session connections (which means that a CCB exists and is chained to a SU's UCB) are received as SPDUs.

The procedure which is performed is always the following:

- The SPDU type and the corresponding connection are identified by a group (A) procedure.
- The SPDU is passed to a group (B) procedure, where a state transition may occur, and the disposition of the SPDU is decided.
- Normally, an event corresponding to the SPDU is notified to the SU by means of a **MESSAGE**, or the SPDU is dropped.

If the SPDU is invalid, then a serious protocol error is detected, in which case no recovery procedure is provided, i.e. the session connection has to be aborted. This uses a variant of the abort procedure described later.

4.5.5 Session connection abortion

Essentially, there may be three interested parties to be notified on abortion of a session connection:

1. The SU, operating the session connection (SU_abort).
2. The remote session entity (abort protocol message).
3. The transport layer.

If Transport generated aborts are treated as a special case for which no SPDU is provided, there is no loss in generality, and the SU may always been alerted.

Another simplification appears if we note that sending an ABORT session protocol message leads to Transport disconnection, and that Transport disconnection, in its turn, leads to abortion at the remote session service.

There are actually two distinct "working" areas for abortion of a session connection.

One is a service, which is available to the session user, and the other is something which is internally generated, due to error conditions. They both follow the general abortion procedure, which is:

- De-allocation of any buffers queued on the CCB (receive and transmit direction, group (A) procedure;

- Sending of the ABORT SPDU supplied by the user (if user generated abort) to the remote session entity, or send the ABORT SPDU supplied by the session entity[7] (if session service generated abort);

- The CCB can not be de-allocated and freed until the transport disconnection is completed (group (C) procedure);

- The local SU is informed about the connection abortion if this was a session generated abort.

On receipt of an ABORT SPDU from the remote session service, the abortion process continues with the following procedure:

- De-allocation of any network buffers queued on the CCB (group (A) procedure).

- Enqueueing of the ABORT.indication on the SU's CCB (group (C) procedure).

- Requesting the disconnection of the Transport connection.

- The CCB can not be deallocated until both the transport disconnection is completed, and the notification of the abort has been passed to the SU by means of a **MESSAGE**.

4.6 PROCEDURES OF GROUP (C)

The procedures of group (C) perform the disposition of the SPDU, and some service functions associated to the SPM behaviour.

The SPDU may have been generated for a Session User. In this case, the arguments for group (C) procedures are:

Connection_identifier

SU-event

The SE-event is either an indication or a confirmation. In any case, the user is alerted by means of a **MESSAGE** function, requested to the Output Queue Handler.

---

[7] The buffer with the incoming SPDU which caused the protocol error is re-used: the session entity overwrites this buffer with the ABORT SPDU.

The SPDU may have been generated to be transferred to the Transport layer. In this case, the arguments for group (C) procedures are:

SPDU address

Transport Layer identifier

Other service functions may be requested for the specified SPDU: for example, the SPDU may be dropped, or replaced by another one (for example, by an ABORT SPDU). The argument for group (C) procedures is, in this case:

Connection_identifier

5.0 MANAGEMENT FUNCTIONS

This section describes the Session Management Functions, which are invoked by the Local Management System. These functions are driven by the **Scheduler**, when events are recognized by the **Input queue handler**.

5.1.1 START event

The START event is generated by the Local Management System when the OSIRIDE Session Layer may start its operations.

5.1.1.1 Parameters

There are no parameters related to this event.

5.1.1.2 Procedure

The originator of the START event is checked. Then:

- If it is not the Local Management System, a MESSAGE is generated for that user, with reason code = Not Authorized.
- If the originator is the Local Management System, the Global State variable is set to ACTIVE, and a MESSAGE is generated. From this moment on, any other event is processed.

5.1.2 STOP event

The STOP event is generated by the Local Management System when the OSIRIDE Session Layer has to immediately stop its operations.

5.1.2.1 Parameters

There are no parameters related to this event.

5.1.2.2 Procedure

The originator of the STOP event is checked. Then:

- If it is not the Local Management System, a MESSAGE is generated for that user, with reason code = Not Authorized.
- If the originator is the Local Management System:

- the Global State variable is set to CFF.
- the users connections are terminated with an ABORT request towards the SPM, and the related Connection Control Blocks (CCB) are freed.
- All users are alerted that their connections have been closed down.
- The users User Control Blocks (UCB) are freed.
- A MESSAGE is generated towards the Local Management System (without parameters). From this moment on, no other event is processed

5.1.3 FLUSH event

The FLUSH event is generated by the Local Management System when the OSIRIDE Session Layer has to softly terminate its operations.

5.1.3.1 Parameters

There are no parameters related to this event.

5.1.3.2 Procedure

The originator of the FLUSH event is checked. Then:

- If it is not the Local Management System, a MESSAGE is generated for that user, with reason code = Not Authorized.
- If the originator is the Local Management System:
  - the Global State variable is set to FLUSHING.
  - A MESSAGE is generated towards the Local Management System (without parameters). From this moment on, no other request for connection is accepted, nor for OPEN, and incoming requests for connection are automatically refused.
  - As soon as the users connections are terminated, the related Connection Control Blocks (CCB) are freed.
  - As soon as no Session connections are active for a user, its User Control Block (UCB) is freed.

5.1.4 STATUS event

The STATUS event is generated by the Local Management System to request information about either the global Session Layer operations, or some particular user's operations.

5.1.4.1 Parameters

What follows is the list of the parameters related to the STATUS event:

- **name**
- **connection id**

5.1.4.2 Procedure

The originator of the STATUS event is checked. Then:

- If it is not the Local Management System, a MESSAGE is generated for that user, with reason code = Not Authorized.
- If the originator is the Local Management System:
  - If the **name** parameter has the value <OSISESSION>, then:
    - A MESSAGE is initiated towards the Local Management System, with parameters:

      Number of users (i.e. number of accepted OPEN operations).

      Name of user A.

      Number of active connections for user A.

      Name of user B.

      Number of active connections for user B.

      ..... (repeat as many times as the number of users)
  - If the **name** parameter has not the value <OSISESSION>:
    - A MESSAGE is initiated towards the Local Management System, with parameters:

      Number of active Session connections for that user.

      Name of the OSIRIDE node related to connection n

      Number of octets sent on connection n

Number of octets received on connection n

Number of resynchronizations on connection n

..... (repeat as many times as the number of active connections)

6.0 SESSION LAYER'S IMPLEMENTATION SPECIFICATIONS

What follows is the detailed, although machine-independent, description of the Session Layer implementation. The description herein is given in terms of a number of routines, which have been written using a pseudo-code, similar to a PASCAL language, but without any particular feature of any existing programming language.

The only assumption we have made is that this pseudo-language is a structured language and, as such, it allows IF, ELSE, ENDIF, DO WHILE and DO UNTIL statements.

All routines are invoked by means of a CALL statement: this presumes that, at least inside the Session Layer, we are in an uni-processor environment, with uni-task structure. Should this not be true in a particular environment, the CALL statements will have to be changed into that environment's peculiar statements, like for instance POST, or AWAKE, or PASS_CONTROL, or FORK, or similar.

From what has been said, it follows that **no Session layer's internal synchronization mechanisms are provided**, and that the Internal Scheduler is a main program rather than a real scheduler. This, in turn, implies that some differencies do exist between this description and the narrative description which has been used so far. Some implementation choices have been taken, which reduce the generality of the preceding description. Thus, what follows is to be intended as an implementation exercise. Real implementations may differ from this particular description.

6.1 DATA STRUCTURES

Two main control blocks are defined in this implementation, which are extensively used in the following sections, that are:

The UCB which describes the Session user. There is one UCB per user, and it is allocated at OPEN time.

The CCB which describes a user's connection. There is one CCB per connection and many CCBs per user (i.e. UCB).

Both the CCb and the UCB bear the user's name.

In addition to CCBs and UCBs, other structures are used in the implementation description:

The Profile is selected at connection establishment time, and brings the default parameters selection for that con-

nection, plus some values which are used for establishing the Transport connection.

The Hostab is searched for at connection establishment phase, and is used to translate from the user given name of the remote system to:

A. A Transport address (TSAP)

B. A network address (X.25 address)

in order for the proper Transport connection be established.

Beside these data structures, a number of queues are used, which are briefly described in the following sections.

6.1.1 The UCB

The User Control Block (UCB) contains all information necessary to identify a user who issued an OPEN request. Among those:

- The name which was declared by the user to identify himself in the OSIRIDE environment.
- The local operating system name, i.e. all information necessary to retrieve the user in the local environment.
- The maximum number of connections that the user is able to request (output connections).
- The maximum number of connections that the user is able to accept (input connections).
- The password(s) to be used to select profiles for given connections at connection establishment time.
- The number of active output connections.
- The number of active input connections.
- The anchor for the chain of the active CCBs (connections).

6.1.2 The CCB

Each CCB describes a user connection, either incoming or outgoing. The CCB contains all information related to a single connection, plus general information, including:

- The remote user address

- The remote system address
- The local user address
- The local user name (same as in the UCB)
- The state of the protocol machine
- The last (or current) outgoing PDU
- The last (or current) incoming PDU
- The input buffer address
- The input buffer length
- The output buffer address
- The output buffer length
- The variables used by the protocol machine
- The list of functional units available for that connection
- The definition and assignement of the tokens

6.1.3 The Profile

The profile is an item contained in a list which is typically stored in mass storage (disk, for instance), and retrieved at Session initialization time. No mechanisms are described here for accessing and loading the profiles' list.

Each item in the profiles' list is called a Profile, and is identified by a letter. A password is related to each profile. Apart form the identifier and the password, a Profile contains:

- Transport related parameters:

  1. Throughput, which will be specified at Transport connection establishment time.

  2. Resilience, which will be specified at Transport connection establishment time.

  3. Transport expedited, which will be requested at Transport connection establishment time (in CSIRIDE Version 1, the Transport expedited is **always** requested).

- Session related parameters:

1. Concatenation, to allow the reception of extended concatenation SPDUs (in OSIRIDE Version 1, extended concatenation is always supported).

2. Functional units list, to be used in that connection.

6.1.4 The Hostab

The Hostab is a list of items which is stored as the Profiles list, and retrieved in the same way. It contains the associations between user's specified nicknames and real X.25 addresses which univoquely identify the OSIRIDE end-systems.

This data structure is logically part of the Transport layer, but is described and dealt with here. It has to be noted that, in different implementations, the same OSIRIDE end-system may have different names, the only thing which really matters being the X.25 address.

6.1.5 The Session queues

Control blocks and data blocks are logically chained to different queues. Although the number and the definition of the Session queues is a typical implementation aspect, some of them will be described in what follows.

The UCB queue When a new UCB is created, it is chained to the UCB queue, which represents the set of all SSAPs. When this queue is empty and the Session layer is in the FLUSHING state, the closing procedures are initiated.

The CCB queues When a new CCB is created, it is chained to the **REQUEST** queue, where it remains until the connection is established. The CCBs are chained also to the **CCB** queue, which represents the set of **all active connections.** When a CCB is dechained from the REQUEST queue, it is chained to the **CCBCHAIN** anchor of the related UCB.

The Data queue When a data block arrives for the SS-user, if the user has not previously issued a RECEIVE operation, the data block is chained to the **Incoming Data** chain anchor of the related CCB. This chain is emptied, one element a time, when the user issues a RECEIVE operation.

The Store queue In certain conditions, the arrival of a certain PDU can not be handled immediately. The PDU is then stored in a queue whose anchor is in the CCB by

means of the ENQUEUE routine (see also "6.7 Not formally described parts" pag. 238), and is retrieved by the RECALL routine when needed.

6.2 INTERNAL SCHEDULER

The Internal Scheduler task is the main program of the Session Layer's implementation, It firstly performs the initialization phase, which consists on allocating a minimal number of data blocks, then sends a MESSAGE to the LMS, and waits for the permission to start operations.

Any subsequent request is then handled and delivered to the appropriate module by the Internal Scheduler.

When no requests are present, the Internal Scheduler puts the entire Session Layer into a WAIT state, which lasts until something arrives, either from the Transport Layer, or by the Session users, or by the LMS.

```
PROCEDURE INTERNAL_SCHEDULER
BEGIN;
  CALL GETBLOCK (UCBLEN)     /* Get the main UCB */
  CALL GETBLOCK (BLOCKLEN) /* Get a data block for that UCB */
  IF  (RETURN EQ 0), THEN
    TEXT := Session Layer Ready
    USER := LMS
    GLOBAL_STATE := OFF
    CALL MESSAGE (USER,TEXT)
    TRUE := TRUE
    DO  WHILE(TRUE)
      CALL WAIT (EXTERNAL_EVENT)
      CALL INPUT_HANDLER
      IF  ((GLOBAL_STATE NE OFF) OR (REQCOMM NE 'START')), THEN
        CASE (REQUSER) OF
          TRANSPORT : CALL LOWER_INTERFACE (REQUEST)
          LMS       : CALL COMMAND_SCANNER (REQUEST)
          ISIDE     : CALL INPUT_SCANNER (REQUEST)
        END_OF_CASE
        CALL DELETE (REQUEST)
      ENDIF
    ENDDO
  ELSE
    TEXT := Session Layer Unable to Setup
    USER := LMS
    CALL MESSAGE (USER,TEXT)
  ENDIF
END of Procedure INTERNAL_SCHEDULER
```

```
PROCEDURE WAIT (EVENT)
BEGIN;
/* Use local Operating System facilities to hold the Session layer task
   until the specified event occurs  */
END of Procedure WAIT
```

6.3 UPPER LAYER INTERFACE

6.3.1 State Tables

The ULI, after the Session Layer is activated, may be in one of the following four states:

1. **OFF**
2. **ACTIVE**
3. **CLOSING**
4. **FLUSHING**

The following events may cause transitions:

- **START** (caused by the Local Management System)
- **STOP** (caused by the Local Management System)
- **FLUSH** (caused by the Local Management System)
- **No Active Connections** (internal event)
- **End of closing operations** (internal event)

The state table described in Fig. 71 pag. 165 is the complete representation of the finite states machine which implements this mechanism.

	OFF	ACTIVE	CLOSING	FLUSHING
START	ACTIVE	ACTIVE	CLOSING	ACTIVE
STOP	OFF	CLOSING	CLOSING	CLOSING
FLUSH	OFF	FLUSHING	CLOSING	FLUSH
No Connect		ACTIVE	CLOSING	CLOSING
End of Operat.			OFF	

Fig. 71. ULI states table

6.3.2 Upper layer Interface routines

The ULI is composed of a number of procedures, some of which constitute the "nucleus" of the Interface, others being only service procedures. These latter are, very often, machine dependent and, as such, are only described by words, or by means of examples.

6.3.2.1 Input Scanner

The input scanner procedure analyzes the request which has been passed to the ULI by the Scheduler.

```
PROCEDURE INPUT_SCANNER (REQUEST CONTROL BLOCK)
BEGIN;
  RETURN := Invalid Operation;
  CASE (OPERATION) OF
    OPEN    : CALL OPEN (USER,NAME,PASSWORD,MAXOUT,MAXIN)
    CLOSE   : CALL CLOSE (USER,NAME)
    READY   : CALL READY (USER,NAME,SYSTEM,USER)
    RECEIVE : CALL RECEIVE (USER,NAME,CONNECTION_ID)
    RESET   : CALL RESET (USER,NAME,OPERATION,CONNECTION_ID)
    CONNECT : CALL CONNECT (USER,NAME,SYSTEM,REMOTE_USER,PROFILE,
                            DATA_TOKEN,MARK_TOKEN,SYNC_TOKEN,
                            TERM_TOKEN,DATA,PASSWORD)
    RESPONSE: CALL RESPONSE (USER,NAME,CONNECTION_ID,OPERATION,
                             CONCATENATION,POINT,TYPE,RESULT,
                             DATA_TOKEN,MARK_TOKEN,SYNC_TOKEN,
                             TERM_TOKEN,RES_DATA,TOK_DATA,MARK_DATA)
    TRANSFER: CALL TRANSFER (USER,NAME,CONNECTION_ID,OPERATION,
                             DATA_TOKEN,MARK_TOKEN,SYNC_TOKEN,
                             TERM_TOKEN,POINT,TYPE,DATA,
                             RES_DATA,TOK_DATA,MARK_DATA)
    ABORT   : CALL ABORT (USER,NAME,CONNECTION_ID,REASON)
    FINISH  : CALL FINISH (USER,NAME,CONNECTION_ID)
  END_OF_CASE
  IF  (OPERATION NE RECEIVE), THEN
    CALL MESSAGE (USER,RETURN)
  ENDIF
  IF  (GLOBAL_STATE EQ CLOSING), THEN
    RETURN := 0
    CALL SEARCH (UCBCHAIN,'NEXT')
    DO WHILE (RETURN EQ 0)
      TEXT := Session Layer Closed
      CALL MESSAGE (UCBUSER,TEXT)
      CALL SEARCH (UCBCHAIN,'NEXT')
    ENDDO
    TEXT := Session Layer De-activated
    CALL MESSAGE ('LMS',TEXT)
    GLOBAL_STATE := OFF
 /* Deallocate memory areas
  ENDIF
END of Procedure INPUT_SCANNER
```

6.3.2.2 Open Procedure

The Open procedure allocates the main user control block (UCB), and fills its fields according to user's supplied parameters.

```
PROCEDURE OPEN (USER,NAME,PASSWORD,MAXOUT,MAXIN)
BEGIN;
  RETURN := Session Terminating;
  IF  (GLOBAL_STATE NE FLUSHING), THEN
    CALL GETBLOCK (UCBLEN)
    IF  (RETURN EQ 0), THEN
      UCBUSER := USER
      UCBNAME := NAME
      UCBPASS := PASSWORD
      UCBMAXOUT := MAXOUT
      UCBMAXIN := MAXIN
      CALL  CHAIN (UCE,UCBCHAIN)
    ENDIF
END of Procedure OPEN
```

6.3.2.3 Close Procedure

The Close procedure provides for an ABORT.request on any active user connection, identified by means of the related control block (CCB). Then, the main user control block (UCB) is dechained and purged.

```
PROCEDURE CLOSE (USER,NAME)
BEGIN;
  RETURN := Open Not Issued;
  CALL SEARCH (UCBCHAIN,NAME)
  IF  (RETURN EQ 0), THEN
    RETURN := Not Authorized
    IF  ((USER EQ UCBUSER) OR (USER EQ 'LMS')), THEN
      RETURN := 0
      CALL LOCK (UCB)
      CALL SEARCH (CCBCHAIN,'NEXT')
      DO WHILE (RETURN EQ 0)
        CALL ABORT (NAME,CONNECTION_ID,'CLOSE')
        CALL SEARCH (CCBCHAIN,'NEXT')
      ENDDO
      RETURN := 0
      CALL DECHAIN (UCE,UCBCHAIN)
      CALL DELETE (UCB)
      USERS_NUMBER := USERS_NUMBER - 1
      IF  ((USERS_NUMBER EQ 0) AND (GLOBAL_STATE EQ FLUSHING), THEN
        GLOBAL_STATE := CLOSING
      ENDIF
    ENDIF
  ENDIF
END of Procedure CLOSE
```

6.3.2.4 Ready Procedure

The Ready procedure sets in the main user control block (UCB) a field which indicates the user READYness to accept incoming connection requests.

```
PROCEDURE READY (USER,NAME,SYSTEM,REMOTE_USER)
BEGIN;
  RETURN := Open Not Issued;
  CALL SEARCH (UCBCHAIN,NAME)
  IF  (RETURN EQ 0), THEN
    CALL LOCK (UCB)
    RETURN := No Input Connections Declared
    IF  (UCBMAXIN GT 0), THEN
      UCBREADY := 'YES'
      UCBSYSTEM := SYSTEM
      UCBUSER := USER
      RETURN := 0
    ENDIF
    CALL UNLOCK (UCB)
  ENDIF
END of Procedure READY
```

6.3.2.5 Receive procedure

The Receive procedure sets in the related connection control block (CCB) a field which indicates the user's willingness to accept incoming data. If data have already arrived, the field is not set, and the user is alerted that data are available.

After that data are passed to the user, the related block is released; then, a new block is requested, and passed to the Transport layer by means of an Enable reception operation. The length of the new block to be allocated is computed according to the needs of the Transport layer at the moment, and **not to the length of the previously released block.** This procedure is repeated any time that a block is released. If no memory is available for the requested length, the Enable reception operation is not performed. This may cause a backpressure on all connections due to a memory shortage provoked by an user not issueing a RECEIVE operation.

```
PROCEDURE RECEIVE (USER,NAME,CONNECTION_ID)
BEGIN;
  RETURN := Open Not Issued;
  CALL SEARCH (UCBCHAIN,NAME)
  IF  (RETURN EQ 0), THEN
    RETURN := Invalid Connection id
    CALL SEARCH (CCECHAIN,CONNECTION_ID)
    IF  (RETURN EQ 0), THEN
      CALL LOCK (CCB)
      RETURN := Data not available
      IF  (CCBDATA EQ 'YES'),THEN
        CALL DECHAIN (CCBINDATA,)
        IF  (LASTBLOCK), THEN
          CCBDATA := 'NO'
        ENDIF
        TEXT := Data available /* Data are passed to the ss user */
        CALL MESSAGE (CCBUSER,TEXT)
        CCBRECEIVE := NO
        CALL DELETE (POINTER)
        NEWLENGTH := CCBTMAX.LENGTH - CCBTDATA_AVAILABLE
        CALL GETBLOCK (NEWLENGTH)
        IF  (RETURN EQ 0), THEN
          CALL ENBLRCV (CCB.REF,POINTER,NEWLENGTH)
          CCBTDATA_AVAILABLE := CCBTMAX.LENGTH
        ENDIF
      ELSE
        CCBRECEIVE := YES
      ENDIF
      CALL  UNLOCK (CCB)
    ENDIF
  ENDIF
END of Procedure RECEIVE
```

6.3.2.6 Reset procedure

The Reset request may be issued for either a READY or a RECEIVE operation: a different subroutine is invoked according to the operation to be RESET.

```
PROCEDURE RESET (USER,NAME,OPERATION,CONNECTION_ID)
BEGIN;
  RETURN := Open Not Issued;
  CALL SEARCH (UCBCHAIN,NAME)
  IF  (RETURN EQ 0), THEN
    RETURN := Invalid Request
    CASE (OPERATION) OF
      RECEIVE : CALL RESREC (UCB,CONNECTION_ID)
      READY   : CALL RESREA (UCB,CONNECTION_ID)
    END_OF_CASE
  ENDIF
END of Procedure RESET
```

```
PROCEDURE RESREC (UCB,CONNECTION_ID)
BEGIN;
  RETURN := Invalid Connection id
  CALL SEARCH (CCBCHAIN,CONNECTION_ID)
  IF  (RETURN EQ 0), THEN
    CALL LOCK (CCB)
    CCBRECEIVE := 'NO'
    CALL UNLOCK (CCB)
  ENDIF
END of Procedure RESREC
```

```
PROCEDURE RESREA (UCB,CONNECTION_ID)
BEGIN;
  CALL LOCK (UCB)
  UCBREADY := 'NO'
  UCBSYSTEM := 0
  UCBUSER := 0
  CALL UNLOCK (UCB)
END of Procedure RESREC
```

6.3.2.7 Connect procedure

The Connect procedure tries to establish a user connection.

```
PROCEDURE CONNECT (USER,NAME,SYSTEM,REMOTE_USER,PROFILE,
                   DATA_TOKEN,MARK_TOKEN,SYNC_TOKEN,
                   TERM_TOKEN,DATA,PASSWORD)
BEGIN;
  RETURN := Session Terminating
  IF  (GLOBAL_STATE NE FLUSHING), THEN
    RETURN := Open not issued
    CALL SEARCH (UCBCHAIN,USER)
    IF  (RETURN EQ 0), THEN
      RETURN := Invalid System
      CALL LOOKUP (HOSTAB,SYSTEM)
      IF  (RETURN EQ 0), THEN
        RETURN := Invalid Profile
        CALL LOOKUP (PROFILETAB,PROFILE)
        IF  (RETURN EQ 0), THEN
          RETURN := Profile Not Authorized
          IF  (UCBPASS EQ PASSWORD(PROFILE)), THEN
            RETURN := Too Many Output Connections
            IF  (UCBACTOUT NE UCBMAXOUT), THEN
              CALL LOCK (UCB)
              CALL GETBLOCK (CCBLEN)
              IF  (RETURN EQ 0), THEN
                CCBTMAX.LENGTH := TMAX (PRCFILE)
                CALL GETBLOCK (CCBMAX.LENGTH)
                IF  (RETURN EQ 0), THEN
                  /* Set the initial values of the
                     appropriate CCB fields, as
                     requested by the ENCODE routine
                     for the CONNECT SPDU. */
                  CCBSTATE := STA1
                  CALL CHAIN (CCB,REQCHAIN)
                  UCBACTOUT := UCBACTOUT + 1
                  CALL OUT_PROT (CN,CCBSTATE)
                ENDIF
              ENDIF
            ENDIF
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of Procedure CONNECT
```

6.3.2.8 Response procedure

The Response request may be issued for the following operations:

CONNECT

SYNC

RESYNCH

FINISH

or combinations of **SYNC, PLEASE** and **GIVE.**

A different subroutine is invoked according to the operation which the RESPONSE is for.

```
PROCEDURE RESPONSE (USER_NAME,CONNECTION_ID,OPERATION,DATA
                    POINT,TYPE,RESULT,DATA,DATA_TOKEN,MARK_TOKEN,
                    SYNC_TOKEN,TERM_TOKEN,TOK_DATA,MARK_DATA,RES_DATA)
BEGIN;
  RETURN := Open Not Issued;
  CALL SEARCH (UCBCHAIN,NAME)
  IF  (RETURN EQ 0), THEN
    RETURN := Invalid Connection id
    CALL SEARCH (CCBCHAIN,CONNECTION_ID)
    IF  (RETURN EQ 0), THEN
      CALL LOCK (CCB)
      CCBOUTDATA := DATA
      CCBDATALEN := LENGTH(DATA)
      CASE (OPERATION) OF
        CONNECT     : CALL RESCON (CCB,RESULT,DATA_TOKEN,MARK_TOKEN,
                                   SYNC_TOKEN,TERM_TOKEN)
        SYNC(only)  : CALL RESSYN (CCB,POINT,TYPE)
        SYNC+GIVE   : CALL RESSYNG (CCB,POINT,TYPE,DATA_TOKEN,MARK_TOKEN,
                                    SYNC_TOKEN,TERM_TOKEN,TOK_DATA)
                                    TOKEN,MARK_TOKEN,
```

```
PROCEDURE RESCON (CCB,RESULT,DATA_TOKEN,MARK_TOKEN,
                  SYNC_TOKEN,TERM_TOKEN)
BEGIN;
  RETURN := Incorrect Data Length
  IF  (CCBDATALEN LE 512), THEN
    IF  (RESULT EQ NEGATIVE), THEN
      CALL OUT_PRCT (RCN,CCB)
      CALL DECHAIN (CCB,CCBCHAIN)
      CALL CHAIN (CCB,FROZCHAIN)
    ELSE
      UCBACTIN := UCBACTIN + 1
      /* Set new proposed values
         for the negotiable parameters,
         to be used by the ENCODE
         routine. */
      CALL OUT_PRCT (RCP,CCB)
    ENDIF
  ENDIF
END of Procedure RESCON
```

```
PROCEDURE RESSYN (CCB,POINT,TYPE,MARK_DATA)
BEGIN;
  RETURN := Incorrect Data Length
  IF  (CCBDATALEN LE 512), THEN
    CCBOUTDATA := MARK_DATA
    RETURN := Wrong Sync Point
    IF  (TYPE EQ MAJOR), THEN
      IF (POINT EQ (CCBVM -1)),THEN
        CCBRESPOINT := POINT
        CALL OUT_PROT(MAA,CCB)
      ENDIF
    ELSE
      IF (TYPE EQ MINOR), THEN
        IF  (CCBVSC EQ TRUE), THEN
          IF  ((POINT LT CCBVM) AND (POINT GE CCBVA)),THEN
            CCBRESPOINT := POINT
            CALL OUT_PROT (MIA,CCB)
          ENDIF
        ENDIF
      ENDIF
  ENDIF
END of Procedure RESSYN
```

```
PROCEDURE RESSYNP (CCB,POINT,TYPE,DATA_TOKEN,MARK_TOKEN,
                   SYNC_TOKEN,TERM_TOKEN,TCK_DATA,MARK_DATA)
BEGIN;
  CCB.CONC := YES
  CCB.DELIVER := NO
  CALL RESSYN (CCB,POINT,TYPE,MARK_DATA)
  IF  (RETURN := 0), THEN
    CCB.DELIVER := YES
    CALL TOKENS (CCB,DATA_TOKEN,MARK_TOKEN,SYNC_TOKEN,TERM_TOKEN,PT)
  ENDIF
  CCB.DELIVER := NO
  CCB.CONC := NO
END of Procedure RESSYNP
```

```
PROCEDURE RESSYNG (CCB,POINT,TYPE,DATA_TOKEN,MARK_TOKEN,
                   SYNC_TOKEN,TERM_TOKEN,TCK_DATA,MARK_DATA)
BEGIN;
  RETURN := Concatenation Not Allowed
  IF  (CCB.CONCAT := YES),THEN
    CCB.CONC := YES
    CCB.DELIVER := NO
    CALL RESSYN (CCB,POINT,TYPE,MARK_DATA)
    IF  (RETURN EQ 0), THEN
      CCB.DELIVER := YES
      CALL TOKENS (DATA_TOKEN,MARK_TOKEN,SYNC_TOKEN,TERM_TOKEN,GT)
      CCB.DELIVER := NO
      CCB.CONC := NO
    ENDIF
  ENDIF
END of Procedure RESSYNG
```

```
PROCEDURE RESRES (CCB,DATA_TOKEN,MARK_TOKEN,
                  SYNC_TOKEN,TERM_TOKEN,POINT)
BEGIN;
  IF  (DATA_TOKEN NE CCB.RES_DK), THEN
    IF  (CCB.RES_DK NE 'CHOICE'), THEN
      RETURN := Incorrect Setting
    ELSE
      CCB.RES_DK := DATA_TOKEN
    ENDIF
  ENDIF
  IF  (RETURN EQ 0), THEN
    IF  (MARK_TOKEN NE CCB.RES_MK), THEN
      IF  (CCB.RES_MK NE 'CHOICE'), THEN
        RETURN := Incorrect Setting
      ELSE
        CCB.RES_MK := MARK_TOKEN
      ENDIF
    ENDIF
    IF  (RETURN EQ 0), THEN
      IF  (SYNC_TOKEN NE CCB.RES_SK), THEN
        IF  (CCB.RES_SK NE 'CHOICE'), THEN
          RETURN := Incorrect Setting
        ELSE
          CCB.RES_SK := SYNC_TOKEN
        ENDIF
      ENDIF
      IF  (RETURN EQ 0), THEN
        IF  (TERM_TOKEN NE CCB.RES_TK), THEN
          IF  (CCB.RES_TK NE 'CHOICE'), THEN
            RETURN := Incorrect Setting
          ELSE
            CCB.RES_TK := TERM_TOKEN
          ENDIF
        ENDIF
        IF  (RETURN EQ 0), THEN
          CCB.RESTYPE := TYPE
          CCBPOINT := POINT
          CALL OUT_PROT (RA,CCB)
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of Procedure RESRES
```

```
PROCEDURE RESFIN (CCB,RESULT)
BEGIN;
  RETURN := Incorrect Data Length
  IF  (CCBDATALEN LE 512), THEN
    RETURN := 0
    IF  (RESULT EQ REJECT), THEN
      FINOUT := FNR
    ELSE
      FINOUT := FNP
    ENDIF
    CALL OUT_PROT ((FINOUT),CCB)
  ENDIF
END of Procedure RESFIN
```

6.3.2.9 Transfer procedure

The Transfer request may be issued for the following operations:

DATA

PLEASE

GIVE

SYNC

RESYNCH

or any combination of **GIVE**, **SYNC** and **DATA**.

A different subroutine is invoked according to the operation which causes TRANSFER of information.

```
PROCEDURE TRANSFER (USER,NAME,CONNECTION_ID,OPERATION,
                    DATA_TOKEN,MARK_TOKEN,SYNC_TOKEN,TERM_TOKEN,
                    POINT,TYPE,DATA,TOK_DATA,RES_DATA,MARK_DATA)
BEGIN;
  RETURN := Open Not Issued;
  CALL SEARCH (UCBCHAIN,NAME)
  IF  (RETURN EQ 0), THEN
    RETURN := Invalid Connection id
    CALL SEARCH (CCBCHAIN,CONNECTION_ID)
    IF  (RETURN EQ 0), THEN
      CALL LOCK (CCB)
      CCBOUTDATA := DATA
      CCBDATALEN := LENGTH(DATA)
      CASE (OPERATION) OF
        DATA(only): CALL OUT_PROT (DR,CCB)
        PLEASE    : CALL TOKENS (CCB,DATA_TOKEN,MARK_TOKEN,
                                 SYNC_TOKEN,TERM_TOKEN,OPERATION)
        GIVE(only): CALL TOKENS (CCB,DATA_TOKEN,MARK_TOKEN,
                                 SYNC_TOKEN,TERM_TOKEN,OPERATION,TOK_DATA
        SYNC(only): CALL SYNC (CCB,POINT,TYPE,MARK_DATA)
        RESYNCH   : CALL RESYNCH (CCB,POINT,TYPE,DATA_TOKEN,MARK_TOKEN,
                                  SYNC_TOKEN,TERM_TOKEN,RES_DATA)
        DATA+GIVE : CALL DATAGIV (CCB,DATA_TOKEN,MARK_TOKEN,
                                  SYNC_TOKEN,TERM_TOKEN,TOK_DATA)
        DATA+SYNC : CALL DATASYN (CCB,POINT,TYPE,MARK_DATA)
        SYNC+GIVE : CALL SYNCGIV (CCB,DATA_TOKEN,MARK_TOKEN,SYNC_TOKEN,
                                  TERM_TOKEN,TOK_DATA,POINT,TYPE,
                                  MARK_DATA)
        DAT+GIV+SYN: CALL DASYGIV (CCB,DATA_TOKEN,MARK_TOKEN,SYNC_TOKEN,
                                   TERM_TOKEN,TOK_DATA,POINT,TYPE,
                                   MARK_DATA)
       END_OF_CASE
       CALL UNLOCK (CCB)
    ENDIF
  ENDIF
END of Procedure TRANSFER
```

```
PROCEDURE DATAGIV (CCB,DATA_TOKEN,MARK_TOKEN,SYNC_TOKEN,
                   TERM_TOKEN,TOK_DATA)
BEGIN;
  RETURN := Invalid data length
  IF  (LENGTH(TOK_DATA) LE 512), THEN
    CCB.CCNC := YES
    CALL OUT_PROT (DR,CCB)
    CCB.DATA_SENT = CCB.DATA_SENT + CCBDATALEN
    CCB.DELIVER := YES
    CALL TOKENS (CCE,DATA_TOKEN,MARK_TOKEN,SYNC_TOKEN,TERM_TOKEN,GIVE,TOK_DATA
    CCB.CONC := NC
    CCB.DELIVER := NO
  ENDIF
END of Procedure DATAGIV
```

```
PROCEDURE SYNCGIV (CCB,DATA_TOKEN,MARK_TOKEN,SYNC_TOKEN,
                   TERM_TOKEN,TOK_DATA,POINT,TYPE,MARK_DATA)
BEGIN;
  RETURN := Invalid data length
  IF  ((LENGTH(MARK_DATA) LE 512) AND (LENGTH(TOK_DATA) LE 512)), THEN
    CCB.CONC := YES
    CALL SYNC (CCB,POINT,TYPE,MARK_DATA)
    CCB.DELIVER := YES
    CALL TOKENS (CCE,DATA_TOKEN,MARK_TOKEN,SYNC_TOKEN,TERM_TOKEN,
                 TOK_DATA)
    CCB.CONC := NC
    CCB.DELIVER := NC
  ENDIF
END of Procedure SYNCGIV
```

```
PROCEDURE DATASYN (CCB,POINT,TYPE,MARK_DATA)
BEGIN;
  RETURN := Concatenation not alllowed;
  IF  (CCBCONCAT EQ YES), THEN
    RETURN := Invalid data length
    IF  (LENGTH(MARK_DATA) LE 512), THEN
      CCB.CONC := YES
      CALL OUT_PROT (DR,CCE)
      CCB.DATA_SENT = CCB.DATA_SENT + CCBDATALEN
      CCB.DELIVER := YES
      CALL SYNC (CCB,PCINT,TYPE,MARK_DATA)
      CCB.CONC := NC
      CCB.DELIVER := NC
    ENDIF
  ENDIF
END of Procedure DATASYN
```

```
PROCEDURE DASYGIV (CCB,POINT,TYPE,MARK_DATA,DATA_TOKEN,MARK_TOKEN,
                   SYNC_TOKEN,TERM_TOKEN,TCK_DATA,DATA)
BEGIN;
  RETURN := Concatenation not alllowed;
  IF  (CCBCONCAT EQ YES), THEN
    RETURN := Invalid data length
    IF  (LENGTH(MARK_DATA) LE 512), THEN
      CCB.CONC := YES
      CALL OUT_PROT (DR,CCE)
      CCB.DATA_SENT = CCB.DATA_SENT + CCB.DATALEN
      CALL SYNC (CCE,POINT,TYPE,MARK_DATA)
      CCB.DELIVER := YES
      CALL TOKENS (CCE,DATA_TOKEN,MARK_TOKEN,SYNC_TOKEN,TERM_TOKEN,GIVE)
      CCB.CONC := NC
      CCB.DELIVER := NC
    ENDIF
  ENDIF
END of Procedure DASYGIV
```

```
PROCEDURE TOKENS (CCB,DATA_TOKEN,MARK_TOKEN,SYNC_TOKEN,
                  TERM_TOKEN,OPERATION,TOK_DATA)
BEGIN;
  RETURN := Invalid Data Length
  IF  (LENGTH(TOK_DATA) LE 512), THEN
    RETURN := 0
    CCBOUTDATA := TOK_DATA
    CCBDATALEN := LENGTH(TOK_DATA)
    /* Set the new tokens setting
       for the protocol routine */
    CALL OUT_PROT ((OPERATION),CCB)
  ENDIF
END of Procedure TOKENS
```

```
PROCEDURE SYNC (CCB,POINT,TYPE)
BEGIN;
    CCBPOINT := POINT
    IF  (CCB.SYNTYPE EQ MAJOR), THEN
      SYNOUT := MAP
    ELSE
      SYNOUT := MIP
    ENDIF
    CALL OUT_PROT ((SYNOUT),CCB)
  ENDIF
END of Procedure SYNC
```

```
PROCEDURE RESYNCH (CCB,POINT,TYPE,DATA_TOKEN,MARK_TOKEN,
                   SYNC_TOKEN,TERM_TOKEN,RES_DATA)
BEGIN;
  RETURN := Invalid Data Length
  IF  (LENGTH(RES_DATA) LE 512), THEN
    CCBPOINT := POINT
    CCB.RESTYPE := TYPE
    CALL OUT_PROT ((CCB.RESTYPE),CCB)
  ENDIF
END of Procedure RESYNCH
```

6.3.2.10 Abort procedure

The Abort abruptly terminates a user connection. The related control block (CCB) is no longer available to the user.

```
PROCEDURE ABORT (USER,NAME,CONNECTION_ID,REASON)
BEGIN;
  RETURN := Open Not Issued;
  CALL SEARCH (UCBCHAIN,NAME)
  IF  (RETURN EQ 0), THEN
    RETURN := Invalid Connection id
    CALL SEARCH (CCBCHAIN,CONNECTION_ID)
    IF  (RETURN EQ 0), THEN
      CCBREASON := REASON
      CCBABORT := YES
      CALL CUT_PROT (AE,CCB)
      CALL  CHAIN (CCB,FRCZCHAIN)
    ENDIF
END of Procedure ABORT
```

6.3.2.11 Finish procedure

The Finish procedure is used to softly terminate the related user connection. No other operations may be issued on that connection before having received a response to that finish.

```
PROCEDURE FINISH (USER,NAME,CONNECTION_ID)
BEGIN;
  RETURN := Open Not Issued;
  CALL SEARCH (UCBCHAIN,NAME)
  IF  (RETURN EQ 0), THEN
    RETURN := Invalid Connection id
    CALL SEARCH (CCBCHAIN,CONNECTION_ID)
    IF  (RETURN EQ 0), THEN
      CCBFINISH := YES
      CALL OUT_PROT (FIN,CCB)
    ENDIF
END of Procedure FINISH
```

6.3.2.12 Getblock Procedure

This is a service routine, Operating system dependent.

```
PROCEDURE GETBLOCK (LENGTH)
BEGIN;
  RETURN := No Memory available
/* Use local Operating System facilities to get a dynamic
   memory area   */
  IF  (Memory has been obtained), THEN
    RETURN := 0
    POINTER := Memory Address
  ENDIF
END of Procedure GETBLOCK
```

6.3.2.13 Delete Procedure

This is a service routine, Operating system dependent.

```
PROCEDURE DELETE (BLOCK)
BEGIN;
/* Use local Operating System facilities to release a dynamic
   memory area obtained by means of a GETBLOCK procedure */
END of Procedure DELETE
```

6.3.2.14 Chain Procedure

This is a service routine, implementation language dependent. The example which is given here assumes that:

- A function exists, called ROOT, which gives the root of a chain, by specifying the chain's name.
- A field exists in any element of the chain, called NEXTAB, which brings the pointer to the next element in the chain, assuming that a variable, called POINTER, points to the current element in the chain.
- As soon as the POINTER variable is set to a specified value, the current block is that one addressed by POINTER.
- The last element in the chain brings, as NEXTAB, the value 0 (zero).

According to the language used for the implementation, however, this routine can be programmed in different ways.

```
PROCEDURE CHAIN (BLOCK_POINTER,CHAIN)
BEGIN;
  POINTER := ROOT (CHAIN)
  DO  WHILE (NEXTAB NE 0)
    PREVIOUS := POINTER
    POINTER := NEXTAB
  ENDDO
  POINTER := PREVIOUS
  NEXTAB := BLOCK_POINTER
  POINTER := BLOCK_POINTER
  NEXTAB := 0
END of Procedure CHAIN
```

6.3.2.15 Dechain Procedure

This is an implementation language dependent routine, which removes an elemnt from a chain.

The example which is given here assumes that:

- Tables have been enchained by means of the CHAIN procedure described in "6.3.2.14 Chain Procedure."
- A function exists, called ROOT, which gives the root of a chain, if the chain's name is specified.
- A field exists in any element of the chain, called NEXTAB, which brings the pointer to the next element in the chain, assuming that a variable, called POINTER, points to the current element in the chain.
- As soon as the POINTER variable is set to a specified value, the current block is that one addressed by POINTER.
- The last element in the chain brings, as NEXTAB, the value 0 (zero).

According to the language used for the implementation, however, this routine can be programmed in different ways.

```
PROCEDURE DECHAIN (CHAIN,ELEMENT)
BEGIN;
  Return := Not found
  POINTER := ROOT(CHAIN)
  FOUND := FALSE
  DO  WHILE ((NEXTAB NE 0) OR (FOUND EQ FALSE))
    IF  (NEXTAB EQ ELEMENT), THEN
      PREVIOUS := POINTER
      POINTER := ELEMENT
      FOLLOWING := NEXTAB
      POINTER := PREVIOUS
      NEXTAB := FOLLOWING
      POINTER := ELEMENT
      FOUND := TRUE
    ENDIF
  ENDDO
  IF  (FOUND EQ TRUE), THEN
    RETURN := 0
  ENDIF
END of Procedure Dechain
```

6.3.2.16 Search Procedure

This is an implementation language dependent routine, which searches in a chain to find either the next element in the chain or an element which brings a specific value in a field.

In the example given in the CHAIN proceduer described in "6.3.2.14 Chain Procedure" pag. 182. Any element in any chain is supposed to carry a pointer to the next element in the chain. However, in order to restrict the implementation language dependent routines to a small number, the function to find the next elemnt in a chain is performed here. This allows to write all other routines independently on the way the chains are implemented. The example which is given here assumes that:

- Tables have been enchained by means of the CHAIN procedure.
- A procedure exists, called LOOKUP, which finds a field in a given table.
- A function exists, called ROOT, which gives the root of a chain, by specifying the chain's name.
- A field exists in any element of the chain, called NEXTAB, which brings the pointer to the next element in the chain, assuming that a variable, called POINTER, points to the current element in the chain.
- As soon as the POINTER variable is set to a specified value, the current block is that one addressed by POINTER.

- The last element in the chain brings, as NEXTAB, the value 0 (zero).

According to the language used for the implementation, however, this routine can be programmed in different ways.

```
PROCEDURE SEARCH (CHAIN,FIELD)
BEGIN;
  Return := Not found
  IF  (FIELD EQ 'NEXT'), THEN
    IF  (NEXTAB NE 0), THEN
      POINTER := NEXTAB
      RETURN := 0
    ENDIF
  ELSE
    POINTER := ROOT(CHAIN)
    FOUND := FALSE
    DO  WHILE ((NEXTAB NE 0) OR (FOUND EQ FALSE))
      POINTER := NEXTAB
      CALL LOOKUP(POINTER,FIELD)
      IF  (RETURN EQ 0), THEN
        FOUND := TRUE
      ENDIF
    ENDDO
    IF  (FOUND EQ TRUE), THEN
      RETURN := 0
    ENDIF
  ENDIF
END of Procedure SEARCH
```

6.3.2.17 Lookup Procedure

This is an implementation language dependent routine, which looks into a table to find a specified field. The example which is given here assumes that:

- Tables are composed of items, and items are composed of fields.
- The end of a table is a condition which is sensed by the programming language.
- A function exists, called NEXT, which gives either the next item in a table, or the next field in an item.

According to the language used for the implementation, however, this routine can be programmed in different ways.

```
PROCEDURE LOOKUP (TABEL,FIELD)
BEGIN;
  FOUND := FALSE
  DO UNTIL (END_OF_TABEL OR FOUND)
    ITEM := NEXT (TABEL)
    DO UNTIL (END_OF_ITEM OR FOUND)
      FIELD(ITEM) := NEXT (FIELD(ITEM))
      IF  (FIELD(ITEM) EQ FIELD), THEN
        FOUND := TRUE
        RETURN := 0
      ENDIF
    ENDDO
  ENDDO
END of Procedure LOOKUP
```

6.4 SESSION PROTOCOL MACHINE

This section contains the detailed description of the SPM. For each valid intersection STATE/EVENT, a routine is provided. These routines corrspond, more or less, to what has been called "Group B procedures" in "4.0 Protocol state machine" pag. 116. Two different handlings of error conditions are described, which correspond to the invalid intersections:

1. Errors provoked by wrong requests/responses, i.e. Session Users errors;

2. Errors provoked by wrong SPDUs (either invalid or not admitted in a given state), i.e. protocol errors.

The SPM implementation has been divided in two parts, which correspond to the calls made to the SPM by either the ULI or the LLI:

1. The **OUTPUT** machine, which describes events coming from the Upper Layer Interface, ULI.

2. The **INPUT** machine, which describes events coming from the Lower Layer Interface, LLI.

6.4.1 OUTPUT machine

The OUTPUT machine correspond to the set of all possible calls from the ULI. For each call, an event is specified, together with a CCB. Apart form the specific event, all information about the parameters, state transition, etc. are found in the CCB.

Some events require that a transmission is required, i.e., a Transport T_DATA.request is invoked. In any case, an unique routine is called, the **SEND** routine. Whether the transmission is actually performed or, as in the case of concatenation, not performed, is a matter of this routine, and **is of no interest for the SPM routine.** The SEND routine is one of those which have been previously called "Group C procedures".

At the end of the protocol operation, if a particular connection (CCB) goes into the state STA1, the de-allocation procedures are initiated for that control block.

```
PROCEDURE OUT_PROT (EVENT,CCB)
  RETURN := 0
  CASE (EVENT) OF
    CN     : CALL CNOUT (CCB)
    RCN    : CALL RFOUT (CCB)
    RCP    : CALL ACOUT (CCB)
    MAA    : CALL MAAOUT (CCB)
    MIA    : CALL MIACOUT (CCB)
    RA     : CALL RAOUT (CCB)
    FNR    : CALL FNROUT (CCB)
    FNP    : CALL FNPOUT (CCB)
    DR     : CALL DROUT (CCB)
    GT     : CALL GTOUT (CCB)
    PT     : CALL PTOUT (CCB)
    MAP    : CALL MAPOUT (CCB)
    MIP    : CALL MIPOUT (CCB)
    ABN    : CALL ABNOUT (CCB)
    RES    : CALL RESOUT (CCB)
    SET    : CALL SETOUT (CCB)
    AB     : CALL ABOUT (CCB)
    FIN    : CALL FINOUT (CCB)
    OTHERWISE : CALL USERR (EVENT,CCB)
  END of CASE
  IF  (CCBSTATE EQ STA1), THEN
    CALL DECHAIN (CCB)
    IF  (CCBVCALL EQ FALSE), THEN
      UCBACTOUT := UCBACTOUT-1
    ELSE
      UCBACTIN := UCBACTIN-1
    ENDIF
    GLOBAL_CONNECTIONS := GLOBAL_CONNECTIONS-1
    IF  ((GLOBAL_STATE EQ FLUSHING) AND (GLOBAL_CONNECTIONS EQ 0)), THEN
      GLOBAL_STATE := CLOSING
    ENDIF
  ENDIF
END of Procedure OUT_PROT
```

6.4.1.1 CN output event

This event may only happen in two states:

1. STA1 (Closed transport not connected)

2. STA1C (Closed transport connected)

```
PROCEDURE DROUT (CCB)
  RETURN := Invalid Request
  IF  ((CCBSTATE EQ STA7/13) OR (CCBSTATE EQ STA10A)), THEN
    IF  ((CCB.DATA EQ 'MINE') OR (CCB.DATA EQ 'NO')), THEN
      RETURN := 0
      CCBOUTPDU := DT
      CALL SEND(CCB)
    ENDIF
  ELSE
    IF  (CCBSTATE EQ STA9), THEN
      IF  ((CCBFUFD EQ TRUE) AND (CCBVCOLL EQ FALSE)), THEN
        RETURN := 0
        CCBOUTPDU := DT
        CALL SEND(CCB)
      ENDIF
    ELSE
      IF (CCBSTATE EQ STA15B), THEN
        RETURN := 0
      ENDIF
    ENDIF
  ENDIF
END of Procedure DROUT
```

6.4.1.10 GT output event

This event may only happen in three states:

1. STA4A (Wait for MAJOR SYNC ACK SPDU)

2. STA7/13 (Session connection is open)

3. STA10A (Wait for S-SYNC-MAJOR.response event)

4. STA15B (After PREPARE, wait for RESYNCHRONIZE)

```
PROCEDURE GTOUT (CCB)
  RETURN := Invalid Request
  IF  (CCB.TOK_DK EQ TRUE), THEN
    IF  (CCB.DATA EQ 'MINE'), THEN
      RETURN := 0
    ENDIF
  ENDIF
  IF  (RETURN EQ 0), THEN
    IF  (CCB.TOK_MK EQ TRUE), THEN
      IF  (CCB.MARK NE 'MINE'), THEN
        RETURN := Invalid Request
      ENDIF
    ENDIF
    IF  (RETURN EQ 0), THEN
      IF  (CCB.TOK_SY EQ TRUE), THEN
        IF  (CCB.SYNC NE 'MINE'), THEN
          RETURN := Invalid Request
        ENDIF
      ENDIF
      IF  (RETURN EQ 0), THEN
        IF  (CCB.TOK_TR EQ TRUE), THEN
          IF  (CCB.TERM NE 'MINE'), THEN
            RETURN := Invalid Request
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDIF
  IF  (RETURN EQ 0), THEN
  RETURN := Invalid Request
    IF  (CCBSTATE EQ (STA4A OR STA7/13 OR STA10A)), THEN
      RETURN := 0
      CCBOUTPDU := GT
      CALL SEND(CCB)
      IF  (CCB.TOK_DK EQ TRUE), THEN
        CCB.DATA := YOURS
      ENDIF
      IF  (CCB.TOK_MK EQ TRUE), THEN
        CCB.MARK := YOURS
      ENDIF
      IF  (CCB.TOK_SY EQ TRUE), THEN
        CCB.SYNC := YOURS
      ENDIF
      IF  (CCB.TOK_TR EQ TRUE), THEN
        CCB.TERM := YOURS
      ENDIF
    ELSE
      IF (CCBSTATE EQ STA15B), THEN
        RETURN := 0
      ENDIF
    ENDIF
  ENDIF
END of Procedure GTOUT
```

6.4.1.11 PT_output_event

This event may only happen in four states:

1. STA7/13 (Session connection is open)

2. STA9 (Wait for S-RELEASE.response event)

3. STA10A (Wait for S-SYNC-MAJOR.response event)

4. STA15B (After PREPARE, wait for RESYNCHRONIZE)

The new tokens assignement is stored in CCB.TOK_DK, CCB.TOK_MK, CCB.TOK_SK and CCB.TOK_TK for Data, Minor, Major and Terminate token respectively.

```
PROCEDURE PTOUT (CCB)
  RETURN := Invalid Request
  IF  (CCB.TOK_DK EQ TRUE), THEN
    IF  (CCB.DATA NE 'NO'), THEN
      RETURN := 0
    ENDIF
  ENDIF
  IF  (RETURN EQ 0), THEN
    IF  (CCB.TOK_MK EQ TRUE), THEN
      IF  (CCB.MARK EQ 'NO'), THEN
        RETURN := Invalid Request
      ENDIF
    ENDIF
    IF  (RETURN EQ 0), THEN
      IF  (CCB.TOK_SY EQ TRUE), THEN
        IF  (CCB.SYNC EQ 'NO'), THEN
          RETURN := Invalid Request
        ENDIF
      ENDIF
      IF  (RETURN EQ 0), THEN
        IF  (CCB.TOK_TR EQ TRUE), THEN
          IF  (CCB.TERM EQ 'NO'), THEN
            RETURN := Invalid Request
          ENDIF
        ENDIF
      ENDIF
    ENDIF
    IF  (RETURN EQ 0), THEN
      RETURN := Invalid Request
        IF  (CCBSTATE EQ (STA7/13 OR STA9 OR STA10A)), THEN
          RETURN := 0
          CCBOUTPDU := PT
          CALL SEND(CCB)
        ENDIF
      ELSE
        IF (CCBSTATE EQ STA15B), THEN
          RETURN := 0
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of Procedure PTOUT
```

6.4.1.12 MAP output event

This event may only happen in two states:

1. STA7/13 (Session connection is open)

2. STA15B (After PREPARE, wait for RESYNCHRONIZE)

```
PROCEDURE MAPOUT (CCB)
  RETURN := Invalid Request
  IF  (CCB.DATA EQ ('MINE' OR 'NC')), THEN
    IF  (CCB.MARK EQ ('MINE' OR 'NO')), THEN
      IF  (CCB.SYNC EQ 'MINE'), THEN
        RETURN := 0
        IF  (CCBSTATE EQ STA7/13), THEN
          CCBOUTPDU := MAP
          CALL SEND(CCB)
          CCBVM := CCBVM + 1
          IF  (CCBVSC EQ TRUE), THEN
            CCBVA := CCBVM
            CCBVSC := FALSE
          ENDIF
        ELSE
          IF (CCBSTATE NE STA15B), THEN
            RETURN := Invalid Request
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of Procedure MAPOUT
```

6.4.1.13 MIP output event

This event may only happen in two states:

1. STA7/13 (Session connection is open)

2. STA15B (After PREPARE, wait for RESYNCHRONIZE)

```
PROCEDURE MIPOUT (CCB)
  RETURN := Invalid Request
  IF  ((CCB.DATA EQ 'MINE') OR (CCB.DATA EQ 'NC'), THEN
    IF  (CCB.MARK EQ 'MINE'), THEN
      RETURN := 0
      IF  (CCBSTATE EQ STA7/13), THEN
        CCBOUTPDU := MIP
        CALL SEND(CCB)
        CCBVM := CCBVM + 1
        IF  (CCBVSC EQ TRUE), THEN
          CCBVA := CCBVM
          CCBVSC := FALSE
        ENDIF
      ELSE
        IF (CCBSTATE NE STA15B), THEN
          RETURN := Invalid Request
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of Procedure MIPOUT
```

6.4.1.14 ABN output event

This event may only happen in seven states:

1. STA4A (Wait for MAJOR SYNC ACK SPDU)
2. STA7/13 (Session connection is open)
3. STA9 (Wait for S-RELEASE.response event)
4. STA10A (Wait for S-SYNC_MAJOR.response event)
5. STA11A (Wait for S-RESYNCHRONIZE.response event)
6. STA15A (After PREPARE, wait for MAJOR SYNC ACK SPDU)
7. STA15B (After PREPARE, wait for RESYNCHRONIZE)

```
PROCEDURE ABNOUT (CCB)
  RETURN := Invalid Request
  IF  (CCBFURS EQ TRUE), THEN
    IF  (CCBSTATE EQ (STA4A OR STA10A)), THEN
      CALL ABSEND
      CCBSTATE := STA5A
    ELSE
      IF  (CCBSTATE EQ STA7/13),THEN
        IF  ((CCBFUACT EQ FALSE) OR (CCBVACT EQ TRUE)),THEN
          CALL ABSEND
          CCBSTATE := STA5A
        ENDIF
      ELSE
        IF  (CCBSTATE EQ STA9), THEN
          IF (CCBFUACT EQ FALSE),THEN
            CALL ABSEND
            CCBSTATE := STA5A
          ENDIF
        ELSE
          IF  (CCBSTATE EQ STA11A), THEN
            RETURN := 0
            CALL RESWINNER (CCB)
            IF  (CCBWIN EQ FALSE), THEN
              CALL ABSEND
              CCBSTATE := STA5A
            ENDIF
          ELSE
            IF (CCBSTATE EQ STA15A), THEN
              IF  ((CCBFUACT EQ FALSE) OR (CCBNEXTACT EQ TRUE)), THEN
                CALL ABSEND
                CCBSTATE := STA5A
              ENDIF
            ELSE
              IF (CCBSTATE EQ STA15B), THEN
                IF  (CCBVRSP EQ 'NO'), THEN
                  CALL ABSEND
                  CCBSTATE := STA6A
                ENDIF
              ENDIF
            ENDIF
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of Procedure ABNOUT
PROCEDURE ABNOUT (CCB)
    RETURN := 0
    CCBOUTPDU := PR-rs
    CALL SEND(CCB)
    CCBOUTPDU := RS-a
    CALL SEND(CCB)
    CCBVRSP := ABD
    CCBVRSPNB := 0
END of Procedure ABNOUT
```

6.4.1.15 RESWINNER procedure

This procedure implements the algorithm necessary to evaluate the function SPM-winner according to what is described in "4.4.2 Definitions" pag. 123.

For the sake of clarity, it is supposed here that the following unequalities apply:

- DISC is greater than INT
- INT is greater than ABD
- ABD is greater than SET
- SET is greater than REST

```
PROCEDURE RESWINNER
  CCBWIN := TRUE
  IF  (CCBVRSP NE 'NO'), THEN
    IF  (CCB.RESTYPE LT CCBVRSP), THEN
      CCBWIN := FALSE
    ELSE
      IF  ((CCB.RESTYPE AND CCBVRSP) EQ 'REST'), THEN
        IF  (CCBPOINT GT CCBVRSPNB), THEN
          CCBWIN := FALSE
        ELSE
          IF  ((CCBPOINT EQ CCBVRSPNB) AND (CCBVTCA EQ FALSE)), THEN
            CCBWIN := FALSE
          ENDIF
        ENDIF
      ELSE
        CCBWIN := FALSE
      ENDIF
    ENDIF
  ENDIF
END of Procedure RESWINNER
```

6.4.1.16 RES output event

This event may only happen in five states:

1. STA7/13 (Session connection is open)
2. STA 9 (Wait for S-RELEASE.response event)
3. STA10A (Wait for S-SYNC_MAJOR.response event)
4. STA11A (Wait for S-RESYNCHRONIZE.response event)
5. STA15B (After PREPARE, wait for RESYNCHRCNIZE)

```
PROCEDURE RESOUT (CCB)
  RETURN := Invalid Request
  IF  (((CCBFUSY OR CCBFUMA) EQ TRUE) AND (CCBFURS EQ TRUE)), THEN
    IF  (CCBPOINT GE CCBVR), THEN
      IF  (CCBSTATE EQ STA7/13), THEN
        IF  ((CCBFUACT EQ FALSE) OR (CCBNEXTACT EQ TRUE)), THEN
          CALL RSSEND
          CCBSTATE := STA5A
        ELSE
          IF  (CCBSTATE EQ STA9), THEN
            IF  (CCBFUACT EQ FALSE), THEN
              CALL RSSEND
              CCBSTATE := STA5A
            ENDIF
          ELSE
            IF  (CCBSTATE EQ STA10A), THEN
              CALL RSSEND
              CCBSTATE := STA5A
            ELSE
              IF  (CCBSTATE EQ STA11A), THEN
                RETURN := 0
                CALL RESWINNER (CCB)
                IF  (CCBWIN EQ FALSE), THEN
                  CALL ABSEND
                  CCBSTATE := STA5A
                ENDIF
              ELSE
                IF (CCBSTATE EQ STA15B), THEN
                  IF  (CCBVRSP EQ 'NO'), THEN
                    CALL RSSEND
                    CCBSTATE := STA6
                  ENDIF
                ENDIF
              ENDIF
            ENDIF
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of Procedure RESOUT
PROCEDURE RSSEND (CCB)
  RETURN := 0
  CCBOUTPDU := PR-rs
  CALL SEND(CCB)
  CCBOUTPDU := RS-r
  CALL SEND(CCB)
  CCBVRSP := RES
  CCBVRSPNB := 0
END of Procedure RSSEND
```

6.4.1.17 SET output event

This event may only happen in six states:

1. STA4A (Wait for MAJOR SYNC ACK SPDU)
2. STA7/13 (Session connection is open)
3. STA9 (Wait for S-RELEASE.response event)
4. STA10A (Wait for S-SYNC_MAJOR.response event)
5. STA11A (Wait for S-RESYNCHRONIZE.response event)
6. STA15B (After PREPARE, wait for RESYNCHRONIZE)

```
PROCEDURE SETOUT (CCB)
  RETURN := Invalid Request
  IF  ((CCBFURS EQ TRUE) AND ((CCBFUACT OR CCBVACT) EQ TRUE)), THEN
    IF  (CCBSTATE EQ (STA4A OR STA10A)), THEN
      CALL NORMSET
      CCBSTATE := STA5A
    ELSE
      IF  (CCBSTATE EQ STA7/13),THEN
        IF  (CCBFUACT EQ FALSE),THEN
          CALL NORMSET
          CCBSTATE := STA5A
        ENDIF
      ELSE
        IF  (CCBSTATE EQ STA9), THEN
          IF ((CCBFUACT EQ FALSE) OR (CCBVACT EQ TRUE)),THEN
            CALL NORMSET
            CCBSTATE := STA5A
          ENDIF
        ELSE
          IF  (CCBSTATE EQ STA11A), THEN
            RETURN := 0
            CALL RESWINNER (CCB)
            IF  (CCBWIN EQ FALSE), THEN
              CALL NORMSET
              CCBSTATE := STA5A
            ENDIF
          ELSE
            IF (CCBSTATE EQ STA15A), THEN
              IF  ((CCBFUACT EQ FALSE) OR (CCBNEXTACT EQ TRUE)), THEN
                CALL NORMSET
                CCBSTATE := STA5A
              ENDIF
            ELSE
              IF (CCBSTATE EQ STA15B), THEN
                IF  (CCBVRSP EQ 'NO'), THEN
                  CALL NORMSET
                  CCBSTATE := STA6
                ENDIF
              ENDIF
            ENDIF
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of Procedure SETOUT
PROCEDURE NORMSET(CCB)
  CCBOUTPDU := PR-rs
  CALL SEND(CCB)
  CCBOUTPDU := RS-s
  CALL SEND(CCB)
  CCBVRSP := SET
  CCBVRSPNB := CCBPOINT
END of Procedure NORMSET
```

6.4.1.18 AB output event

This event may happen in all states except:

1. STA1C (Closed transport connected)

2. STA16 (Wait T-DISCONNECT event)

```
PROCEDURE ABOUT (CCE)
  RETURN := Invalid Request
  IF  ((CCBSTATE NE (STA1C OR STA16)), THEN
    RETURN := 0
    IF  (CCBSTATE EQ STA1B), THEN
      CALL T_DISC (CCB)
      CCBSTATE := STA1
    ELSE
      CCBOUTPDU := AE-nr
      CALL SEND(CCB)
      CCBSTATE := STA16
    ENDIF
  ENDIF
END of Procedure ABOUT
```

6.4.1.19 FIN output event

This event may only happen in three states:

1. STA7/13 (Session connection is open)

2. STA9 (Wait for S-RELEASE.response event)

3. STA15B (After PREPARE, wait for RESYNCHRONIZE)

```
PROCEDURE SETOUT (CCB)
  RETURN := Invalid Request
  IF  ((CCBFURS EQ TRUE) AND ((CCBFUACT OR CCBVACT) EQ TRUE)), THEN
    IF  (CCBSTATE EQ (STA4A OR STA10A)), THEN
      CALL NORMSET
      CCBSTATE := STA5A
    ELSE
      IF  (CCBSTATE EQ STA7/13),THEN
        IF  (CCBFUACT EQ FALSE),THEN
          CALL NORMSET
          CCBSTATE := STA5A
        ENDIF
      ELSE
        IF  (CCBSTATE EQ STA9), THEN
          IF ((CCBFUACT EQ FALSE) OR (CCBVACT EQ TRUE)),THEN
            CALL NORMSET
            CCBSTATE := STA5A
          ENDIF
        ELSE
          IF  (CCBSTATE EQ STA11A), THEN
            RETURN := 0
            CALL RESWINNER (CCB)
            IF  (CCBWIN EQ FALSE), THEN
              CALL NORMSET
              CCBSTATE := STA5A
            ENDIF
          ELSE
            IF (CCBSTATE EQ STA15A), THEN
              IF  ((CCBFUACT EQ FALSE) OR (CCBNEXTACT EQ TRUE)), THEN
                CALL NORMSET
                CCBSTATE := STA5A
              ENDIF
            ELSE
              IF (CCBSTATE EQ STA15B), THEN
                IF  (CCBVRSP EQ 'NO'), THEN
                  CALL NORMSET
                  CCBSTATE := STA6
                ENDIF
              ENDIF
            ENDIF
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of Procedure SETOUT
PROCEDURE NORMSET(CCB)
  CCBOUTPDU := PR-rs
  CALL SEND(CCB)
  CCBOUTPDU := RS-s
  CALL SEND(CCB)
  CCBVRSP := SET
  CCBVRSPNB := CCBPOINT
END of Procedure NORMSET
```

6.4.1.18 AB output event

This event may happen in all states except:

1. STA1C (Closed transport connected)

2. STA16 (Wait T-DISCONNECT event)

```
PROCEDURE ABOUT (CCE)
  RETURN := Invalid Request
  IF  ((CCBSTATE NE (STA1C OR STA16)), THEN
    RETURN := 0
    IF  (CCBSTATE EQ STA1B), THEN
      CALL T_DISC (CCB)
      CCBSTATE := STA1
    ELSE
      CCBOUTPDU := AE-nr
      CALL SEND(CCB)
      CCBSTATE := STA16
    ENDIF
  ENDIF
END of Procedure ABOUT
```

6.4.1.19 FIN output event

This event may only happen in three states:

1. STA7/13 (Session connection is open)

2. STA9 (Wait for S-RELEASE.response event)

3. STA15B (After PREPARE, wait for RESYNCHRONIZE)

```
PROCEDURE FINOUT (CCB)
  RETURN := Invalid Request
  IF  (CCBSTATE EQ (STA7/13 OR STA15B)), THEN
    IF  ((CCB.DATA EQ ('MINE' OR 'NO')), THEN
      IF  ((CCB.MARK EQ ('MINE' OR 'NO')), THEN
        IF  ((CCB.SYNC EQ ('MINE' OR 'NO')), THEN
          IF  ((CCB.TERM EQ ('MINE' OR 'NO')), THEN
            RETURN := 0
            IF  (CCBSTATE EQ STA7/13), THEN
              CCBOUTPDU := FN-nr
              CALL SEND (CCB)
              CCBVTRR := FALSE
              CCBSTATE := STA3
            ENDIF
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ELSE
    IF  (CCBSTATE EQ STA9), THEN
      IF  ((CCB.DATA AND CCB.MARK AND CCB.SYNC AND CCB.TERM) EQ 'NO),THEN
        RETURN := 0
        CCBOUTPDU := FN-nr
        CALL SEND(CCB)
        CCBVTRR := FALSE
        CCBVCOLL := TRUE
        CCBSTATE := STA9
      ENDIF
    ENDIF
  ENDIF
END of Procedure FINOUT
```

6.4.1.20 SEND Procedure

The SEND procedure performs the transfer of Protocol Data Units by means of a T-DATA.request.

The actual transfer (i.e. invoking the T-DATA.request primitive) may not happen in case of concatenation. The decision is based on the two variables CONC and DELIVER which are defined in the CCB.

As the encoding algorithm of a PDU depends on the characteristics of the implementation language, no explicit description is given here of how the encoding is performed. However, all necessary information is contained in the CCB before any encoding operation is started. The CONC variable is used by the encoding routine to decide how to structure the buffer which has to be delivered to the transport layer. The ENCODE routine cannot be described here, its characteristics being dependent on the implementation language. Hower, the encoding algorithm is described in "3.4.4 Concatenation" pag. 25 of "First Part." For each legal SPDU, the ENCODE routine uses certain fields of the CCB related to the connection. The list of valid SPDUs, fields and CCB values is given in "6.7 Not formally described parts" pag. 238.

It is assumed here that, after the encoding routine has been called, the field **CCBTRANS** contains the address of the buffer to be delivered to the transport layer, and **TRANSLEN** contains the length of that buffer.

```
PROCEDURE SEND
  CALL ENCODE (CCB)
  IF  (CCBOUTPDU EQ ( PR OR AB )), THEN
    CALL T_EXDATA (CCB.TCEP,CCB.REF,CCB.TRANS,TRANSLEN)
  ELSE
    IF  ((CONC EQ 'NO') OR (DELIVER EQ 'YES')), THEN
      CALL T_DATA (CCB.TCEP,CCB.REF,CCB.TRANS,TRANSLEN)
    ENDIF
  ENDIF
END of PROCEDURE SEND
```

6.4.2 INPUT machine

The INPUT machine corresponds to the set of all possible calls to the SPM by the LLI (see also "6.5 Lower layer interface" pag. 229). For each call, an event is specified, together with a CCB. Apart from the specific event, which is indicated in the call, all information about the parameters, state, etc. are supposedly found in the CCB.

Most of the events caused by the arrival of a Protocol Data Unit (PDU) require an indication to be delivered to the Session user. This is done by means of a call to the MESSAGE routine, and by indicating in the text of the message the type of indication to be delivered. It is supposed that all those parameters which pertain to a given indication are actually delivered to the SS-user. The list of parameters contained in any indication is specified in "2.0 Session Service" pag. 7 in "First Part."

When the protocol message processing is terminated, if the state of the connection is STA1, then the related CCB is de-allocated. Then, if this means that the last session connection has been closed, and the GLOBAL_STATE variable indicates **FLUSHING**, the closing procedures are initiated, the Local Management System is alerted, and the session layer closes operations.

```
PROCEDURE IN_PROT (EVENT,CCB)
  CASE (EVENT) OF
    AB : CALL ABIN(CCB)
    AA : CALL ABIN(CCB)
    AC : CALL ACIN(CCB)
    CN : CALL CNIN(CCB)
    DT : CALL DTIN(CCB)
    DN : CALL DNIN(CCB)
    FN : CALL FNIN(CCB)
    GT : CALL GTIN(CCB)
    MAA : CALL MAAIN(CCB)
    MAP : CALL MAPIN(CCB)
    MIA : CALL MIAIN(CCB)
    MIP : CALL MIPIN(CCB)
    NF : CALL NFIN(CCB)
    PT : CALL PTIN(CCB)
    PR : CALL PRIN(CCB)
    RA : CALL RAIN(CCB)
    RS : CALL RSIN(CCB)
 END OF CASE
 IF  (CCBSTATE EQ STA1), THEN
   CALL DECHAIN (CCB)
   IF  (CCBVCALL EQ FALSE), THEN
     UCBACTOUT := UCBACTOUT-1
   ELSE
     UCBACTIN := UCBACTIN-1
   ENDIF
   GLOBAL_CONNECTIONS := GLOBAL_CONNECTIONS-1
   IF  ((GLOBAL_STATE EQ FLUSHING) AND (GLOBAL_CONNECTIONS EQ 0)), THEN
     GLOBAL_STATE := CLOSING
     RETURN := 0
     CALL SEARCH (UCBCHAIN,'NEXT')
     DO WHILE (RETURN EQ 0)
       TEXT := Session Layer Closed
       CALL MESSAGE (UCBUSER,TEXT)
       CALL SEARCH (UCBCHAIN,'NEXT')
     ENDDO
     TEXT := Session Layer De-activated
     CALL MESSAGE ('LMS',TEXT)
     GLOBAL_STATE := OFF
/* Deallocate memory areas
   ENDIF
  ENDIF
END of PROCEDURE IN_PROT
```

6.4.2.1 AB procedure

This procedure is invoked when either an Abort PDU or an Abort Accept PDU is received.

The reception of this PDU is valid in all states.

```
PROCEDURE ABIN(CCB)
  CALL T_DISC (CCB)
  CCBSTATE := STA1
  IF  (CCBINPDU EQ AB), THEN
    IF  (CCBSTATE NE (STA1C OR STA16)), THEN
      TEXT := Abort Indication
      CALL MESSAGE (CCBUSER,TEXT)
    ENDIF
  ENDIF
END of PROCEDURE ABIN
```

6.4.2.2 AC procedure

This procedure is invoked when an Accept PDU is received. All parameters related to the session connection are stored in the CCB.

The reception of this PDU is only valid in the following states:

- STA1C (Idle Transport Connected)
- STA2A (Wait for ACCEPT SPDU)
- STA16 (Wait for T-disconnect.indication)

```
  PROCEDURE ACIN(CCB)
    IF  (CCBSTATE EQ STA2A), THEN
      TEXT := Connect Confirmation
      CALL MESSAGE (CCBUSER,TEXT)
      CCBVA := CCB.IN_NUMBER
      CCBVM := CCBVA
      CCBVCALL := FALSE
      CCBVRSP := NO
      CCBSTATE := STA7/13
    ELSE
      IF  (CCBSTATE EQ STA1C), THEN
        CALL TDISC
        CCBSTATE := STA1
      ELSE
        IF  (CCBSTATE NE STA16), THEN
          CALL PROTER (CCE)
        ENDIF
      ENDIF
    ENDIF
  END of PROCEDURE ACIN
```

```
PROCEDURE PROTER (CCB)
  CCB.REASON := Protocol Error
  CALL OUT_PROT (AB,CCE)
  CCBSTATE := STA16
  CALL DECHAIN (CCB)
  CALL CHAIN (CCB,FROZCHAIN)
END of Procedure PROTER
```

6.4.2.3 CN procedure

This procedure is invoked when a Connect PDU is received. All parameters related to the session connection are stored in the CCB.

The reception of this PDU is only valid in the following states:

- STA1C (Idle Transport connected)
- STA16 (Wait for T-disconnect.indication)

```
PROCEDURE CNIN(CCB)
  IF  (CCBSTATE EQ STA1C), THEN
    IF  (CCBVTCA EQ TRUE), THEN
      CALL SEARCH (UCBCHAIN,CCBUSER)
      IF  (RETURN EQ 0), THEN
        CALL DECHAIN (INCHAIN,CCB)
        CALL CHAIN (CCBCHAIN(UCB),CCB)
        TEXT := Connect Indication
        CALL MESSAGE (CCBUSER,TEXT)
        CCBSTATE := STA8
      ELSE
        CCBOUTPDU := RF-nr
        CCBTDATA := User not connected at SSAP
        CALL SEND (CCB)
        CCBSTATE := STA16
      ENDIF
    ELSE
      CALL T_DISC (CCB)
      CCBSTATE := STA1
    ENDIF
  ELSE
    IF  (CCBSTATE NE STA16), THEN
      CALL PROCTERR (CCB)
    ELSE
      CALL T_DISC (CCB)
      CCBSTATE := STA1
    ENDIF
  ENDIF
END of PROCEDURE CNIN
```

6.4.2.4 DT procedure

This procedure is invoked when a Data PDU is received. All parameters related to the session connection are stored in the CCB.

Incoming data are addressed by the NEWDATA field of the CCB. Data are passed directly to the SS user if the CCBRECEIVE field is set to 'YES'. Otherwise they are enqueued in the CCBINDATA chain. In case of error, data are purged.

The reception of this PDU is only valid in the following states:

- STA1C (Idle Transport Connected)
- STA3 (Wait for DISCONNECT SPDU)
- STA4A (Wait for MAJOR SYNC CONFIRMATION SPDU)
- STA5A (Wait for RESYNCHRONIZE ACK SPDU - no collision)
- STA6 (Wait for RESYNCHRONIZE ACK SPDU -collision)
- STA7/13 (Session connection is open)
- STA15A (After PREPARE, wait for MAJOR SYNC CONFIRMATION SPDU)
- STA15B (After PREPARE, wait for RESYNCHRONIZE SPDU)
- STA16 (Wait for T-disconnect.indication)

```
EDURE DTIN(CCB)
  (CCBSTATE EQ (STA4A OR STA15A)), THEN
IF  (CCBFUFD EQ TRUE), THEN
  IF  (CCBRECEIVE EQ 'NO'), THEN
    CALL CHAIN (NEWDATA,CCBINDATA)
    TEXT := Incoming Data
    CALL MESSAGE (CCBUSER,TEXT)
  ELSE
    TEXT := Data available /* Data are passed to the ss user */
    CALL MESSAGE (CCBUSER,TEXT)
    CCBRECEIVE := NO
    CALL DELETE (NEWDATA)
    NEWLENGTH := CCBTMAX.LENGTH - CCBTDATA_AVAILABLE
    CALL GETBLOCK (NEWLENGTH)
    IF  (RETURN EQ 0), THEN
      CALL ENBLRCV (CCB.REF,POINTER,NEWLENGTH)
      CCBTDATA_AVAILABLE := CCBTMAX.LENGTH
    ENDIF
  ENDIF
ELSE
  CALL PROTER (CCB)
ENDIF
SE
IF  (CCB.DATA NE 'MINE'), THEN
  IF  (((CCBSTATE EQ STA3) AND (CCBVCOLL EQ FALSE)) OR (CCBSTATE EQ STA7/13)),
    IF  (CCBRECEIVE EQ 'NO'), THEN
      CALL CHAIN (NEWDATA,CCBINDATA)
      TEXT := Incoming Data
      CALL MESSAGE (CCBUSER,TEXT)
    ELSE
      TEXT := Data available /* Data are passed to the ss user */
      CALL MESSAGE (CCBUSER,TEXT)
      CCBRECEIVE := NO
      CALL DELETE (NEWDATA)
      NEWLENGTH := CCBTMAX.LENGTH - CCBTDATA_AVAILABLE
      CALL GETBLOCK (NEWLENGTH)
      IF  (RETURN EQ 0), THEN
        CALL ENBLRCV (CCB.REF,POINTER,NEWLENGTH)
        CCBTDATA_AVAILABLE := CCBTMAX.LENGTH
      ENDIF
    ENDIF
  ELSE
    IF  (CCBSTATE NE (STA5A OR STA6 OR STA15B OR STA16)), THEN
      CALL PROTER (CCB)
    ENDIF
  ENDIF
ELSE
  CALL PROTER (CCB)
ENDIF
NDIF
of PROCEDURE DTIN
```

6.4.2.5 DN procedure

This procedure is invoked when a Disconnect PDU is received.

The reception of this PDU is only valid in the following states:

- STA1C (Closed transport connected)
- STA3 (Wait for DISCONNECT SPDU)
- STA9 (Wait for S-RELEASE.response)
- STA16 (Wait for T-disconnect.indication)

```
PROCEDURE DNIN(CCB)
  IF  (CCBSTATE EQ STA1C), THEN
    CALL T_DISC (CCB)
    CCBSTATE := STA1
  ELSE
    IF  (CCBSTATE EQ STA3), THEN
      TEXT := Release Confirmation Positive
      CALL MESSAGE (CCBUSER,TEXT)
      IF  (CCBVTRR EQ FALSE), THEN
        CALL T_DISC (CCB)
        CCBSTATE := STA1
      ELSE
        CCBSTATE := STA1C
      ENDIF
    ELSE
      IF  (CCBSTATE EQ STA9), THEN
        IF  (CCBVCALL EQ TRUE), THEN
          TEXT := Release Confirmation Positive
          CALL MESSAGE (CCBUSER,TEXT)
        ELSE
          CALL PROTER(CCB)
        ENDIF
      ELSE
        IF  (CCBSTATE NE STA16), THEN
          CALL PROTER(CCB)
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of PROCEDURE DNIN
```

6.4.2.6 FN procedure

This procedure is invoked when a Finish PDU is received.

The reception of this PDU is only valid in the following states:

- STA1C (Idle Transport Connected)

```
EDURE DTIN(CCB)
  (CCBSTATE EQ (STA4A OR STA15A)), THEN
IF  (CCBFUFD EQ TRUE), THEN
  IF  (CCBRECEIVE EQ 'NO'), THEN
    CALL CHAIN (NEWDATA,CCBINDATA)
    TEXT := Incoming Data
    CALL MESSAGE (CCBUSER,TEXT)
  ELSE
    TEXT := Data available /* Data are passed to the ss user */
    CALL MESSAGE (CCBUSER,TEXT)
    CCBRECEIVE := NO
    CALL DELETE (NEWDATA)
    NEWLENGTH := CCBTMAX.LENGTH - CCBTDATA_AVAILABLE
    CALL GETBLOCK (NEWLENGTH)
    IF  (RETURN EQ 0), THEN
      CALL ENBLRCV (CCB.REF,POINTER,NEWLENGTH)
      CCBTDATA_AVAILABLE := CCBTMAX.LENGTH
    ENDIF
  ENDIF
ELSE
  CALL PROTER (CCB)
ENDIF
SE
IF  (CCB.DATA NE 'MINE'), THEN
  IF  (((CCBSTATE EQ STA3) AND (CCBVCOLL EQ FALSE)) OR (CCBSTATE EQ STA7/13)),
    IF  (CCBRECEIVE EQ 'NO'), THEN
      CALL CHAIN (NEWDATA,CCBINDATA)
      TEXT := Incoming Data
      CALL MESSAGE (CCBUSER,TEXT)
    ELSE
      TEXT := Data available /* Data are passed to the ss user */
      CALL MESSAGE (CCBUSER,TEXT)
      CCBRECEIVE := NO
      CALL DELETE (NEWDATA)
      NEWLENGTH := CCBTMAX.LENGTH - CCBTDATA_AVAILABLE
      CALL GETBLOCK (NEWLENGTH)
      IF  (RETURN EQ 0), THEN
        CALL ENBLRCV (CCB.REF,POINTER,NEWLENGTH)
        CCBTDATA_AVAILABLE := CCBTMAX.LENGTH
      ENDIF
    ENDIF
  ELSE
    IF  (CCBSTATE NE (STA5A OR STA6 OR STA15B OR STA16)), THEN
      CALL PROTER (CCB)
    ENDIF
  ENDIF
ELSE
  CALL PROTER (CCB)
ENDIF
NDIF
of PROCEDURE DTIN
```

6.4.2.5 DN procedure

This procedure is invoked when a Disconnect PDU is received.

The reception of this PDU is only valid in the following states:

- STA1C (Closed transport connected)
- STA3 (Wait for DISCONNECT SPDU)
- STA9 (Wait for S-RELEASE.response)
- STA16 (Wait for T-disconnect.indication)

```
PROCEDURE DNIN(CCB)
  IF  (CCBSTATE EQ STA1C), THEN
    CALL T_DISC (CCB)
    CCBSTATE := STA1
  ELSE
    IF  (CCBSTATE EQ STA3), THEN
      TEXT := Release Confirmation Positive
      CALL MESSAGE (CCBUSER,TEXT)
      IF  (CCBVTRR EQ FALSE), THEN
        CALL T_DISC (CCB)
        CCBSTATE := STA1
      ELSE
        CCBSTATE := STA1C
      ENDIF
    ELSE
      IF  (CCBSTATE EQ STA9), THEN
        IF  (CCBVCALL EQ TRUE), THEN
          TEXT := Release Confirmation Positive
          CALL MESSAGE (CCBUSER,TEXT)
        ELSE
          CALL PROTER(CCB)
        ENDIF
      ELSE
        IF  (CCBSTATE NE STA16), THEN
          CALL PROTER(CCB)
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of PROCEDURE DNIN
```

6.4.2.6 FN procedure

This procedure is invoked when a Finish PDU is received.

The reception of this PDU is only valid in the following states:

- STA1C (Idle Transport Connected)

```
PROCEDURE GTIN(CCB)
  IF  (CCBSTATE NE (STA1C OR STA16), THEN
    IF  (NOT((CCB.DATA NE YOURS) AND (CCB.IN_DT EQ TRUE))), THEN
      IF  (NOT((CCB.MARK NE YOURS) AND (CCB.IN_MK EQ TRUE))), THEN
        IF  (NOT((CCB.SYNC NE YOURS) AND (CCB.IN_SY EQ TRUE))), THEN
          IF  (NOT((CCB.TERM NE YOURS) AND (CCB.IN_TR EQ TRUE))), THEN
            IF  (CCBSTATE EQ (STA4A OR STA7/13 OR STA 15A)), THEN
              TEXT := Give Tokens Indication
              CALL MESSAGE (CCBUSER,TEXT)
              IF  (CCB.IN_DT EQ TRUE), THEN
                CCB.DATA := 'MINE'
              ENDIF
              IF  (CCB.IN_MK EQ TRUE), THEN
                CCB.MARK := 'MINE'
              ENDIF
              IF  (CCB.IN_SY EQ TRUE), THEN
                CCB.SYNC := 'MINE'
              ENDIF
              IF  (CCB.IN_TR EQ TRUE), THEN
                CCB.TERM := 'MINE'
              ENDIF
            ELSE
              IF  (CCBSTATE NE (STA5A OR STA6 OR STA15B OR STA15C)), THEN
                CALL PROTER (CCB)
              ENDIF
            ENDIF
          ELSE
            CALL PROTER (CCB)
          ENDIF
        ELSE
          CALL PROTER (CCB)
        ENDIF
      ELSE
        CALL PROTER (CCB)
      ENDIF
    ELSE
      CALL PROTER (CCB)
    ENDIF
  ELSE
    IF  (CCBSTATE EQ STA1C), THEN
      CALL T_DISC (CCB)
      CCBSTATE := STA1
    ENDIF
  ENDIF
END of PROCEDURE GTIN
```

6.4.2.8 MAA procedure

This procedure is invoked when a Major Ack PDU is received.

The reception of this PDU is only valid in the following states:

- STA4A (Wait for MAJOR SYNC CONFIRMATION SPDU)
- STA5A (Wait for RESYNCHRONIZE ACK SPDU - no collision)
- STA6 (Wait for RESYNCHRONIZE ACK SPDU -collision)
- STA15A (After PREPARE, wait for MAJOR SYNC CONFIRMATION SPDU)
- STA15B (After PREPARE, wait for RESYNCHRONIZE SPDU)
- STA15C (After PREPARE, wait for RESYNCHRONIZE ACK SPDU)
- STA16 (Wait for T-disconnect.indication)

```
PROCEDURE MAAIN(CCB)
  IF  (CCBSTATE NE (STA5A OR STA6 OR STA15B OR STA15C OR STA16)), THEN
    IF  (CCBPOINT EQ (CCBVM - 1)), THEN
      IF  (CCBSTATE EQ STA4A), THEN
        IF  (CCB.TREX EQ FALSE), THEN
          TEXT := Sync Major Confirmation
          CALL MESSAGE (CCBUSER,TEXT)
          CCBVACT := CCBVNEXTACT
          CCBVR := CCBVM
          CCBVA := CCBVM
          CCBSTATE := STA7/13
        ELSE
          CALL PROTER(CCB)
        ENDIF
      ELSE
        IF  (CCBSTATE EQ STA15A), THEN
          IF  ((CCBFUACT EQ FALSE) OR (CCBVNEXTACT EQ TRUE)), THEN
            TEXT := Sync Major Confirmation
            CALL MESSAGE (CCBUSER,TEXT)
            CCBVACT := CCBVNEXTACT
            CCBVR := CCBVM
            CCBVA := CCBVM
            CCBSTATE := STA7/13
            CALL RECALL (CCB)
          ELSE
            CALL PROTER (CCB)
          ENDIF
        ELSE
          CALL PROTER (CCB)
        ENDIF
      ELSE
        CALL PROTER (CCB)
      ENDIF
    ENDIF
  ENDIF
END of PROCEDURE MAAIN
```

```
PROCEDURE GTIN(CCB)
  IF  (CCBSTATE NE (STA1C OR STA16), THEN
    IF  (NOT((CCB.DATA NE YOURS) AND (CCB.IN_DT EQ TRUE))), THEN
      IF  (NOT((CCB.MARK NE YOURS) AND (CCB.IN_MK EQ TRUE))), THEN
        IF  (NOT((CCB.SYNC NE YOURS) AND (CCB.IN_SY EQ TRUE))), THEN
          IF  (NOT((CCB.TERM NE YOURS) AND (CCB.IN_TR EQ TRUE))), THEN
            IF  (CCBSTATE EQ (STA4A OR STA7/13 OR STA 15A)), THEN
              TEXT := Give Tokens Indication
              CALL MESSAGE (CCBUSER,TEXT)
              IF  (CCB.IN_DT EQ TRUE), THEN
                CCB.DATA := 'MINE'
              ENDIF
              IF  (CCB.IN_MK EQ TRUE), THEN
                CCB.MARK := 'MINE'
              ENDIF
              IF  (CCB.IN_SY EQ TRUE), THEN
                CCB.SYNC := 'MINE'
              ENDIF
              IF  (CCB.IN_TR EQ TRUE), THEN
                CCB.TERM := 'MINE'
              ENDIF
            ELSE
              IF  (CCBSTATE NE (STA5A OR STA6 OR STA15B OR STA15C)), THEN
                CALL PROTER (CCB)
              ENDIF
            ENDIF
          ELSE
            CALL PROTER (CCB)
          ENDIF
        ELSE
          CALL PROTER (CCB)
        ENDIF
      ELSE
        CALL PROTER (CCB)
      ENDIF
    ELSE
      CALL PROTER (CCB)
    ENDIF
  ELSE
    IF  (CCBSTATE EQ STA1C), THEN
      CALL T_DISC (CCB)
      CCBSTATE := STA1
    ENDIF
  ENDIF
END of PROCEDURE GTIN
```

6.4.2.8 MAA procedure

This procedure is invoked when a Major Ack PDU is received.

The reception of this PDU is only valid in the following states:

- STA4A (Wait for MAJOR SYNC CONFIRMATION SPDU)
- STA5A (Wait for RESYNCHRONIZE ACK SPDU - no collision)
- STA6 (Wait for RESYNCHRONIZE ACK SPDU -collision)
- STA15A (After PREPARE, wait for MAJOR SYNC CONFIRMATION SPDU)
- STA15B (After PREPARE, wait for RESYNCHRONIZE SPDU)
- STA15C (After PREPARE, wait for RESYNCHRONIZE ACK SPDU)
- STA16 (Wait for T-disconnect.indication)

```
PROCEDURE MAAIN(CCB)
  IF  (CCBSTATE NE (STA5A OR STA6 OR STA15B OR STA15C OR STA16)), THEN
    IF  (CCBPCINT EQ (CCBVM - 1)), THEN
      IF  (CCBSTATE EQ STA4A), THEN
        IF  (CCB.TREX EQ FALSE), THEN
          TEXT := Sync Major Confirmation
          CALL MESSAGE (CCBUSER,TEXT)
          CCBVACT := CCBVNEXTACT
          CCBVR := CCBVM
          CCBVA := CCBVM
          CCBSTATE := STA7/13
        ELSE
          CALL PROTER(CCB)
        ENDIF
      ELSE
        IF  (CCBSTATE EQ STA15A), THEN
          IF  ((CCBFUACT EQ FALSE) OR (CCBVNEXTACT EQ TRUE)), THEN
            TEXT := Sync Major Confirmation
            CALL MESSAGE (CCBUSER,TEXT)
            CCBVACT := CCBVNEXTACT
            CCBVR := CCBVM
            CCBVA := CCBVM
            CCBSTATE := STA7/13
            CALL RECALL (CCB)
          ELSE
            CALL PROTER (CCB)
          ENDIF
        ELSE
          CALL PROTER (CCB)
        ENDIF
      ELSE
        CALL PROTER (CCB)
      ENDIF
    ENDIF
  ENDIF
END of PROCEDURE MAAIN
```

6.4.2.9 MAP procedure

This procedure is invoked when a Major Sync PDU is received.

The reception of this PDU is only valid in the following states:

- STA5A (Wait for RESYNCHRONIZE ACK SPDU - no collision)
- STA6 (Wait for RESYNCHRONIZE ACK SPDU -collision)
- STA7/13 (Session connection is open)
- STA15B (After PREPARE, wait for RESYNCHRONIZE SPDU)
- STA15C (After PREPARE, wait for RESYNCHRONIZE ACK SPDU)
- STA16 (Wait for T-disconnect.indication)

```
PROCEDURE MAPIN(CCB)
  IF  (CCBSTATE NE STA16), THEN
    IF  ((CCB.DATA AND CCB.MARK AND CCB.SYNC) NE 'NO'), THEN
      IF  ((CCBFUACT EQ FALSE) OR (CCBVACT EQ TRUE)), THEN
        IF  (CCBSTATE NE (STA5A OR STA6 OR STA15B OR STA15C)), THEN
          IF  (CCBSTATE EQ STA7/13), THEN
            IF  (CCBPOINT EQ CCBVM), THEN
              TEXT := Sync Major Indication
              CALL MESSAGE (CCBUSER,TEXT)
              IF  (CCBFUACT EQ TRUE), THEN
                CCBVNEXTACT := TRUE
              ENDIF
              CCBVM := CCBVM + 1
              IF  (CCBVSC EQ FALSE), THEN
                CCBVA := CCBVM
              ENDIF
              CCBSTATE := STA10A
            ELSE
              CALL PROTER(CCB)
            ENDIF
          ELSE
            CALL PROTER(CCB)
          ENDIF
        ENDIF
      ELSE
        CALL PROTER(CCB)
      ENDIF
    ELSE
      CALL PROTER(CCB)
    ENDIF
  ENDIF
END of PROCEDURE MAPIN
```

5.4.2.10 MIA procedure

This procedure is invoked when a Minor Ack PDU is received.

The reception of this PDU is only valid in the following states:

- STA3 (Wait for DISCONNECT SPDU)
- STA4A (Wait for MAJOR SYNC CONFIRMATION SPDU)
- STA5A (Wait for RESYNCHRONIZE ACK SPDU - no collision)
- STA6 (Wait for RESYNCHRONIZE ACK SPDU -collision)
- STA7/13 (Session connection is open)
- STA15A (After PREPARE, wait for MAJOR SYNC CONFIRMATION SPDU)
- STA15B (After PREPARE, wait for RESYNCHRONIZE SPDU)
- STA15C (After PREPARE, wait for RESYNCHRONIZE ACK SPDU)
- STA16 (Wait for T-disconnect.indication)

```
PROCEDURE MIAIN(CCB)
  IF  (CCBSTATE NE STA16), THEN
    IF  ((CCB.FU_SY EQ TRUE) AND (CCBVSC EQ FALSE)), THEN
      IF  ((CCBFUACT EQ FALSE) OR (CCBVACT EQ TRUE)), THEN
        IF  (CCBSTATE EQ (STA3 OR STA4A OR STA7/13 OR STA15A)), THEN
          IF  ((CCBPOINT GT CCBVM) AND (CCBPOINT LE CCBVA)), THEN
            TEXT := Sync Minor Confirmation
            CALL MESSAGE (CCBUSER,TEXT)
            CCBVA := CCBPOINT + 1
          ELSE
            CALL PROTER (CCB)
          ENDIF
        ELSE
          IF  (CCBSTATE NE (STA5A OR STA6 OR STA15B OR STA15C)), THEN
            CALL PROTER (CCB)
          ENDIF
        ENDIF
      ELSE
        CALL PROTER (CCB)
      ENDIF
    ELSE
      CAL PROTER (CCB)
    ENDIF
  ENDIF
END of PROCEDURE MIAIN
```

6.4.2.11 MIP procedure

This procedure is invoked when a Minor Sync PDU is received.

The reception of this PDU is only valid in the following states:

- STA5A (Wait for RESYNCHRONIZE ACK SPDU - no collision)
- STA6 (Wait for RESYNCHRONIZE ACK SPDU -collision)
- STA7/13 (Session connection is open)
- STA15B (After PREPARE, wait for RESYNCHRONIZE SPDU)
- STA15C (After PREPARE, wait for RESYNCHRONIZE ACK SPDU)
- STA16 (Wait for T-disconnect.indication)

```
PROCEDURE MIPIN(CCB)
  IF  (CCBSTATE NE STA16), THEN
    IF  ((CCB.DATA NE 'MINE') AND (CCB.SYNC EQ 'YOURS')), THEN
      IF  ((CCBFUACT EQ FALSE) OR (CCBVACT EQ TRUE)), THEN
        IF  (CCBSTATE EQ STA7/13), THEN
          IF  (CCBPOINT EQ CCBVM), THEN
            TEXT := Sync Minor Indication
            CALL MESSAGE (CCBUSER,TEXT)
            CCBVM := CCBVM + 1
            IF  (CCBVSC EQ FALSE), THEN
              CCBVA := CCBVM
              CCBVSC := TRUE
            ENDIF
          ELSE
            CALL PROTER (CCB)
          ENDIF
        ELSE
          IF  (CCBSTATE NE (STA5A OR STA6 OR STA15B OR STA15C)), THEN
            CALL PROTER (CCB)
          ENDIF
        ENDIF
      ELSE
        CALL PROTER (CCB)
      ENDIF
    ELSE
      CALL PROTER (CCB)
    ENDIF
  ENDIF
END of PROCEDURE MIPIN
```

6.4.2.12 NF procedure

This procedure is invoked when a Not Finished PDU is received.

The reception of this PDU is only valid in the following states:

- STA1C (Closed transport connected)
- STA3 (Wait for DISCONNECT SPDU)
- STA15B (After PREPARE, wait for RESYNCHRONIZE SPDU)
- STA16 (Wait for T-disconnect.indication)

```
PROCEDURE NFIN(CCB)
  IF  (CCBSTATE NE STA16), THEN
    IF  (CCBSTATE EQ STA1C), THEN
      CALL T_DISC (CCB)
      CCBSTATE := STA1
    ELSE
      IF  (CCB.FU_TR EQ TRUE), THEN
        IF  (CCBSTATE EQ (STA3 OR STA15B)), THEN
          TEXT := Release Confirmation Negative
          CALL MESSAGE (USER,TEXT)
          IF  (CCBSTATE EQ STA3), THEN
            CCBSTATE := STA7/13
          ENDIF
        ELSE
          CALL PROTER (CCB)
        ENDIF
      ELSE
        CALL PROTER (CCB)
      ENDIF
    ENDIF
  ENDIF
END of PROCEDURE NFIN
```

6.4.2.13 PT procedure

This procedure is invoked when a Please Tokens PDU is received.

The reception of this PDU is only valid in the following states:

- STA1C (Idle Transport Connected)
- STA3 (Wait for DISCONNECT SPDU)
- STA4A (Wait for MAJOR SYNC CONFIRMATION SPDU)
- STA5A (Wait for RESYNCHRONIZE ACK SPDU - no collision)
- STA6 (Wait for RESYNCHRONIZE ACK SPDU -collision)

STA7/13 (Session connection is open)

STA15A (After PREPARE, wait for MAJOR SYNC CONFIRMATION SPDU)

STA15B (After PREPARE, wait for RESYNCHRONIZE SPDU)

STA15B (After PREPARE, wait for RESYNCHRONIZE ACK SPDU)

STA16 (Wait for T-disconnect.indication)

```
PROCEDURE PTIN(CCB)
  IF  (CCBSTATE NE STA16), THEN
    IF  (CCBSTATE EQ STA1C), THEN
      CALL T_DISC (CCB)
      CCBSTATE := STA1
    ELSE
      IF  (NOT((CCB.DATA EQ NO) AND (CCB.IN_DT EQ TRUE))), THEN
        IF  (NOT((CCB.MARK EQ NO) AND (CCB.IN_MK EQ TRUE))), THEN
          IF  (NOT((CCB.SYNC EQ NO) AND (CCB.IN_SY EQ TRUE))), THEN
            IF  (NOT((CCB.TERM EQ NO) AND (CCB.IN_TR EQ TRUE))), THEN
              IF  (CCBSTATE EQ (STA3 OR STA4A OR STA7/13 OR STA15A)), THEN
                TEXT := Please Tokens Indication
                CALL MESSAGE (CCBUSER,TEXT)
              ELSE
                IF  (CCBSTATE NE (STA5A OR STA6 OR STA15B OR STA15C)), THEN
                  CALL  PROTER (CCB)
                ENDIF
              ENDIF
            ELSE
              CALL PROTER (CCB)
            ENDIF
          ELSE
            CALL PROTER (CCB)
          ENDIF
        ELSE
          CALL PROTER (CCB)
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of PROCEDURE PTIN
```

4.2.14 PR procedure

This procedure is invoked when a Prepare PDU is received.

The reception of this PDU is only valid in the following states:

STA1C (Closed transport connected)

- STA2A (Wait for ACCEPT SPDU)
- STA3 (Wait for DISCONNECT SPDU)
- STA4A (Wait for MAJOR SYNC CONFIRMATION SPDU)
- STA5A (Wait for RESYNCHRONIZE ACK SPDU - no collision)
- STA6 (Wait for RESYNCHRONIZE ACK SPDU -collision)
- STA7/13 (Session connection is open)
- STA10A (Wait for S-SYNC_MAJOR.response)
- STA15B (After PREPARE, wait for RESYNCHRONIZE SPDU)
- STA16 (Wait for T-disconnect.indication)

```
PROCEDURE PRIN(CCB)
  CASE (CCBTYPE) OF
    MCD : CALL PRMCD (CCB)
    RA  : CALL PRRA (CCB)
    RS  : CALL PRRS (CCB)
  OTHERWISE: CALL PROTER (CBB)
  END OF CASE
END of Procedure PRIN (CCB)
```

```
PROCEDURE PRMCD (CCB)
  IF  (CCBSTATE NE (STA5A OR STA16)), THEN
    IF  (CCBSTATE EQ STA4A), THEN
      CCBSTATE := STA15A
    ELSE
      IF  (CCBSTATE EQ STA1C), THEN
        CALL TDISC
        CCBSTATE := STA1
      ELSE
        CALL PROTER (CCB)
      ENDIF
    ENDIF
  ENDIF
END of Procedure PRMCD
```

```
PROCEDURE PRRA (CCB)
  IF  (CCBSTATE NE STA16), THEN
    IF  (CCBSTATE EQ STA5A), THEN
      CCBSTATE := STA15C
    ELSE
      IF  (CCBSTATE EQ STA6), THEN
        CALL ENQUEUE (CCB)
      ELSE
        CALL PROTER (CCB)
      ENDIF
    ENDIF
  ENDIF
END of Procedure PRRA



PROCEDURE PRRS (CCB)
  IF  (CCBSTATE EQ (STA3 OR STA4A OR STA7/13 OR STA10A)), THEN
    CCBSTATE := STA15B
  ELSE
    IF  (CCBSTATE EQ STA5A), THEN
      CCBSTATE := STA6
    ELSE
      IF  (CCBSTATE EQ (STA2A OR STA6 OR STA15A)), THEN
        CALL ENQUEUE (CCB)
      ELSE
        IF  (CCBSTATE EQ STA15C), THEN
          CALL CLEARQ (CCB)
          CCBSTATE := STA15B
        ELSE
          CALL PROTER (CCB)
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of PROCEDURE PRRA
```

6.4.2.15 RA procedure

This procedure is invoked when a Resync Ack PDU is received.

The reception of this PDU is only valid in the following states:

- STA5A (Wait for RESYNCHRONIZE ACK SPDU - no collision)
- STA15B (After PREPARE, wait for RESYNCHRONIZE SPDU)
- STA15C (After PREPARE, wait for RESYNCHRONIZE ACK SPDU)
- STA16 (Wait for T-disconnect.indication)

```
PROCEDURE RAIN(CCB)
  IF  (CCBSTATE NE (STA15B OR STA16)), THEN
    IF  ((CCBVRSP EQ RST) AND (CCBPOINT EQ CCBVRSPNB)), THEN
      IF  (CCBSTATE EQ STA15C), THEN
        IF  ((CCB.FU_RS EQ TRUE) AND (CCB.TR_EX EQ TRUE)), THEN
          TEXT := Resync Confirmation
          CALL MESSAGE (CCBUSER,TEXT)
          CCBVM := CCBPOINT
          CCBVA := CCBVM
          IF  (CCBVRSP EQ (ABD OR SET)), THEN
            CCBVR := 0
          ENDIF
          CCBVRSP := NO
          CCBSTATE := STA7/13
          CALL RECALL (CCB)
        ELSE
          CALL PROTER (CCB)
        ENDIF
      ELSE
        IF  (CCBSTATE EQ STA5A), THEN
          IF  ((CCB.FU_RS EQ TRUE) AND (CCB.TR_EX NE TRUE)), THEN
            TEXT := Resync Confirmation
            CALL MESSAGE (CCBUSER,TEXT)
            CCBVM := CCBPOINT
            CCBVA := CCBVM
            IF  (CCBVRSP EQ (ABD OR SET)), THEN
              CCBVR := 0
            ENDIF
            CCBVRSP := NO
            CCBSTATE := STA7/13
            CALL RECALL (CCB)
          ELSE
            CALL PROTER (CCB)
          ENDIF
        ELSE
          CALL PROTER (CCB)
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of PROCEDURE RAIN
```

6.4.2.16 RS procedure

This procedure is invoked when a Resync PDU is received.

The reception of this PDU is only valid in the following states:

- STA3 (Wait for DISCONNECT SPDU)
- STA4A (Wait for MAJOR SYNC CONFIRMATION SPDU)

```
PROCEDURE PRRA (CCB)
  IF  (CCBSTATE NE STA16), THEN
    IF  (CCBSTATE EQ STA5A), THEN
      CCBSTATE := STA15C
    ELSE
      IF  (CCBSTATE EQ STA6), THEN
        CALL ENQUEUE (CCB)
      ELSE
        CALL PROTER (CCB)
      ENDIF
    ENDIF
  ENDIF
END of Procedure PRRA



PROCEDURE PRRS (CCB)
  IF  (CCBSTATE EQ (STA3 OR STA4A OR STA7/13 OR STA10A)), THEN
    CCBSTATE := STA15B
  ELSE
    IF  (CCBSTATE EQ STA5A), THEN
      CCBSTATE := STA6
    ELSE
      IF  (CCBSTATE EQ (STA2A OR STA6 OR STA15A)), THEN
        CALL ENQUEUE (CCB)
      ELSE
        IF  (CCBSTATE EQ STA15C), THEN
          CALL CLEARQ (CCB)
          CCBSTATE := STA15B
        ELSE
          CALL PROTER (CCB)
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of PROCEDURE PRRA
```

6.4.2.15 RA procedure

This procedure is invoked when a Resync Ack PDU is received.

The reception of this PDU is only valid in the following states:

- STA5A (Wait for RESYNCHRONIZE ACK SPDU - no collision)
- STA15B (After PREPARE, wait for RESYNCHRONIZE SPDU)
- STA15C (After PREPARE, wait for RESYNCHRONIZE ACK SPDU)
- STA16 (Wait for T-disconnect.indication)

```
PROCEDURE RAIN(CCB)
  IF  (CCBSTATE NE (STA15B OR STA16)), THEN
    IF  ((CCBVRSP EQ RST) AND (CCBPOINT EQ CCBVRSPNB)), THEN
      IF  (CCBSTATE EQ STA15C), THEN
        IF  ((CCB.FU_RS EQ TRUE) AND (CCB.TR_EX EQ TRUE)), THEN
          TEXT := Resync Confirmation
          CALL MESSAGE (CCBUSER,TEXT)
          CCBVM := CCBPOINT
          CCBVA := CCBVM
          IF  (CCBVRSP EQ (ABD OR SET)), THEN
            CCBVR := 0
          ENDIF
          CCBVRSP := NO
          CCBSTATE := STA7/13
          CALL RECALL (CCB)
        ELSE
          CALL PROTER (CCB)
        ENDIF
      ELSE
        IF  (CCBSTATE EQ STA5A), THEN
          IF  ((CCB.FU_RS EQ TRUE) AND (CCB.TR_EX NE TRUE)), THEN
            TEXT := Resync Confirmation
            CALL MESSAGE (CCBUSER,TEXT)
            CCBVM := CCBPOINT
            CCBVA := CCBVM
            IF  (CCBVRSP EQ (ABD OR SET)), THEN
              CCBVR := 0
            ENDIF
            CCBVRSP := NO
            CCBSTATE := STA7/13
            CALL RECALL (CCB)
          ELSE
            CALL PROTER (CCB)
          ENDIF
        ELSE
          CALL PROTER (CCB)
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of PROCEDURE RAIN
```

6.4.2.16 RS procedure

This procedure is invoked when a Resync PDU is received.

The reception of this PDU is only valid in the following states:

- STA3 (Wait for DISCONNECT SPDU)
- STA4A (Wait for MAJOR SYNC CONFIRMATION SPDU)

- STA5A (Wait for RESYNCHRONIZE ACK SPDU - no collision)
- STA6 (Wait for RESYNCHRONIZE ACK SPDU -collision)
- STA7/13 (Session connection is open)
- STA10A (Wait for S-SYNC_MAJOR.response)
- STA15B (After PREPARE, wait for RESYNCHRONIZE SPDU)
- STA16 (Wait for T-disconnect.indication)

```
PROCEDURE RSIN(CCB)
  IF  (CCBSTATE NE STA16), THEN
    CALL RESWINNER (CCB)
    CASE (CCB.RESTYPE) OF
      RST : CALL RS_RST (CCB)
      ABD : CALL RS_ABD (CCB)
      SET : CALL RS_SET (CCB)
      OTHERWISE : CALL PROTER (CCB)
    END OF CASE
  ENDIF
END of PROCEDURE RSIN
```

```
PROCEDURE RS_RST (CCB)
  IF  (CCBPOINT GE CCBVB), THEN
    IF  (CCBSTATE EQ (STA3 OR STA3A)), THEN
      IF  (((CCB.FU_RS EQ TRUE) AND (CCB.TR_EX EQ TRUE)) AND ((CCBSTATE EQ STA3) OR (CCBFUACT NE TRUE))), THEN
        CALL RSACT (CCB)
      ELSE
        CALL PROTER (CCB)
      ENDIF
    ELSE
      IF  (CCBSTATE EQ STA5A), THEN
        IF  (CCBFURS EQ TRUE) AND (CCB.TR_EX EQ FALSE)), THEN
          CALL RESWINNER
          IF  (CCBWIN EQ FALSE), THEN
            CALL RSACT (CCB)
          ENDIF
        ELSE
          CALL PROTER (CCB)
        ENDIF
      ELSE
        IF  (CCBSTATE EQ STA6), THEN
          CALL RESWINNER
          IF  (CCBWIN EQ TRUE), THEN
            CCBSTATE := STA5A
            CALL RECALL (CCB)
          ELSE
            CALL RSACT (CCB)
            CCBSTATE := STA11A
            CALL RECALL (CCB)
          ENDIF
        ELSE
          IF  (CCBSTATE EQ STA7/13), THEN
            IF  (((CCB.FU_RS EQ TRUE) AND (CCB.TR_EX EQ FALSE)) AND ((CCBFUACT EQ FALSE) OR (CCBVACT EQ TRUE))
                CCBSTATE := STA11A
            ELSE
              CALL PROTER (CCB)
            ENDIF
          ELSE
            IF  (CCBSTATE EQ STA15B), THEN
              IF  (((CCBFUACT EQ FALSE) OR CCBVACT EQ TRUE)) AND (CCBFURS EQ TRUE)), THEN
                CALL RSACT (CCB)
              ELSE
                CALL PROTER (CCB)
              ENDIF
            ELSE
              CALL PROTER (CCB)
            ENDIF
          ENDIF
        ENDIF
      ENDIF
    ENDIF
  ENDIF
END of PROCEDURE RS_RST
```

```
PROCEDURE RSACT (CCE)
  TEXT := Resync Indication
  CALL MESSAGE (CCBUSER,TEXT)
  CCBVRSP := CCBTYPE
  CCBVRSPNB := CCBPCINT
  CCBSTATE := STA11A
END of PROCEDURE RSACT
```

```
PROCEDURE RS_ABD (CCB)
  IF  (CCBSTATE EQ (STA3 OR STA4A OR STA5A OR STA7/13 OR STA10A)), THEN
    IF  ((CCB.FU_RS EQ TRUE) AND (CCB.TE_EX EQ FALSE)), THEN
      IF  (CCBSTATE EQ (STA10A OR STA4A)), THEN
        CALL RSACT (CCB)
      ELSE
        IF  ((CCBSTATE EQ STA3) AND (CCBFUACT EQ FALSE)), THEN
          CCBVM := MAXIMUM(CCBVM,CCBPOINT)
          CALL RSACT (CCB)
        ELSE
          IF  (CCBSTATE EQ STA5A), THEN
            CALL RESWINNER (CCB)
            IF  (CCBWIN EQ FALSE), THEN
              CCBVM := MAXIMUM(CCBVM,CCBPOINT)
              CALL RSACT (CCB)
            ENDIF
          ELSE
            IF  ((CCBSTATE EQ STA7/13) AND ((CCBFUACT EQ FALSE) OR (CCBVACT EQ TRUE))), THEN
              CALL RSACT (CCB)
            ELSE
              CALL PROTER (CCB)
            ENDIF
          ENDIF
        ENDIF
      ENDIF
    ELSE
     CALL PROTER (CCB)
    ENDIF
  ELSE
    IF  (CCBSTATE EQ STA6), THEN
      CALL RESWINNER (CCB)
      IF  (CCBWIN EQ TRUE), THEN
        CCBSTATE := STA5A
        CALL RECALL (CCB)
      ELSE
        CALL RSACT (CCB)
        CALL RECALL (CCB)
      ENDIF
    ELSE
      IF  (CCBSTATE EQ STA15B), THEN
        IF  ((CCB.FU_RS EQ TRUE) AND ((CCBFUACT EQ FALSE) OR (CCBVACT EQ TRUE))), THEN
          CCBVM := MAXIMUM(CCBVM,CCBPOINT)
          CALL RSACT (CCB)
        ELSE
          CALL PROTER (CCB)
        ENDIF
      ELSE
        CALL PROTER (CCB)
      ENDIF
    ENDIF
  ENDIF
END of PROCEDURE RS_ABD
```

```
PROCEDURE RS_SET (CCB)
  IF  (CCBSTATE EQ (STA3 OR STA4A OR STA5A OR STA7/13 OR STA10A)), THEN
    IF  ((CCB.FU_RS EQ TRUE) AND (CCB.TB_EX EQ FALSE)), THEN
      IF  (CCBSTATE EQ (STA4A OR STA10A)), THEN
        CALL RSACT (CCB)
      ELSE
        IF  ((CCBSTATE EQ STA3) AND (CCBFUACT EQ FALSE)), THEN
          CALL RSACT (CCB)
        ELSE
          IF  (CCBSTATE EQ STA5A), THEN
            CALL RESWINNER (CCB)
            IF  (CCBWIN EQ FALSE), THEN
              CCBVM := MAXIMUM(CCBVM,CCBPOINT)
              CALL RSACT (CCB)
            ENDIF
          ELSE
            IF  ((CCBSTATE EQ STA7/13) AND ((CCBFUACT EQ FALSE) OR (CCBVACT EQ TRUE))), THEN
              CALL RSACT (CCB)
            ELSE
              CALL PRCTER (CCB)
            ENDIF
          ENDIF
        ENDIF
      ENDIF
    ELSE
     CALL PRCTER (CCB)
    ENDIF
  ELSE
    IF  (CCBSTATE EQ STA6), THEN
      CALL RESWINNER (CCB)
      IF  (CCBWIN EQ TRUE), THEN
        CCBSTATE := STA5A
        CALL RECALL (CCB)
      ELSE
        CALL RSACT (CCB)
        CALL RECALL (CCB)
      ENDIF
    ELSE
      IF  (CCBSTATE EQ STA15B), THEN
        IF  ((CCB.FU_RS EQ TRUE) AND ((CCBFUACT EQ FALSE) OR (CCBVACT EQ TRUE))), THEN
          CALL RSACT (CCB)
        ELSE
          CALL PRCTER (CCB)
        ENDIF
      ELSE
        CALL PRCTER (CCB)
      ENDIF
    ENDIF
  ENDIF
END of PROCEDURE RS_SET
```

Space

6.5 LOWER LAYER INTERFACE

6.5.1 State Tables

The same finite states machine which applies in the case of the Upper Layer Interface is used for the Lower Layer Interface. This state machine is described in Fig. 71 pag. 165.

When in state **ACTIVE**, and only in that state, the Lower Layer Interface is enabled to work.

For each connection (CCB), another state machine is defined for the LLI, which is described in Fig. 72 pag. 230. The states and events present in that table reflect the state of the underlying Transport connection.

The LLI, after a the Session layer has been activated, for each Session connection may be in one of the following states:

1. **OFF**
2. **ACTIVATING**
3. **CONNECTED**
4. **CLOSING**

The following events may cause transitions:

- **CONNECT SPDU** (caused by the Session Protocol Machine)
- **DISCONNECT SPDU** (caused by the Session Protocol Machine)
- **ABORT SPDU** (caused by the Session Protocol Machine)
- **T-CONNECT.confirm** (coming from the Transport layer)
- **T-CONNECT.indication** (coming from the Transport layer)
- **T-DISCONNECT.indication** (coming from the Transport layer)

	OFF	ACTIVATING	CONNECTED	CLOSING
CONNECT SPDU	T-CONNECT .req [2] ACTIVATING			
DISCONN SPDU			CLOSING	
ABORT SPDU		T-DISC .req CLOSING	CLOSING	T-DISC .req CLOSING
T-CONNECT .confirm	ACTIVE	send CONNECT SPDU	CLOSING	
T-CONNECT .ind		T-CONNECT .resp ACTIVE	OFF	
T-DISC .ind		Generate ABORT.ind OFF	Generate ABORT.ind OFF	CFF

Fig. 72. LLI states table

6.5.2 Lower layer interface routines

What follows is an implementation description of the LLI routines described in "3.0 Interface logical structure" pag. 85. The functions of the LLI are:

- To handle incoming events from the Transport Layer.
- To separate the possibly concatenated PDUs.
- To set the appropriate CCB fields according to the PDU parameters.

6.5.2.1 Procedure Lower Interface

This procedure represent the LLI internal scheduler.

If an error is recognized, the Local Management System is alerted.

At the end of the operations, the state of the connection (CCB) is checked: if the CCBSTATE is STA1, the control block is de-allocated and the closing procedures are initiated.

```
PROCEDURE LOWER_INTERFACE (REQUEST)
BEGIN;
  RETURN := Invalid Operation;
  CASE (OPERATION) OF
    CONN_IND  : CALL CONNIND (REFERENCE,QOS,REM_ADDRESS)
    CONN_CONF : CALL CONCONF (REFERENCE,QOS,TCEP_ID)
    DISC_IND  : CALL DISCIND (REFERENCE,REASON)
    DATA_IND  : CALL DATAIND (REFERENCE,DATA_POINTER,DATA_LENGTH)
    EXPD_IND  : CALL DATAIND (REFERENCE,DATA_POINTER,DATA_LENGTH)
    ENBL_TRX  : CALL ENBLTRX (REFERENCE,DATA_POINTER,DATA_LENGTH)
  END_OF_CASE
  IF  (RETURN NE 0), THEN
    MESSAGE ('LMS',RETURN)
  ENDIF
  IF  (CCBSTATE EQ STA1), THEN
    CALL DECHAIN (CCB)
    IF  (CCBVCALL EQ FALSE), THEN
      UCBACTOUT := UCBACTOUT-1
    ELSE
      UCBACTIN := UCBACTIN-1
    ENDIF
    GLOBAL_CONNECTIONS := GLOBAL_CONNECTIONS-1
    IF  ((GLOBAL_STATE EQ FLUSHING) AND (GLOBAL_CONNECTIONS EQ 0)), THEN
      GLOBAL_STATE := CLOSING
      RETURN := 0
      CALL SEARCH (UCBCHAIN,'NEXT')
      DO WHILE (RETURN EQ 0)
        TEXT := Session Layer Closed
        CALL MESSAGE (UCBUSER,TEXT)
        CALL SEARCH (UCBCHAIN,'NEXT')
      ENDDO
      TEXT := Session Layer De-activated
      CALL MESSAGE ('LMS',TEXT)
      GLOBAL_STATE := OFF
 /* Deallocate memory areas
     ENDIF
   ENDIF
END of Procedure Lower Interface
```

6.5.2.2 T-connect.indication procedure

A T-connect.indication in OSIRIDE is refused if and only if no memory resources are available to satisfy another Session connection.

```
PROCEDURE CONNIND (REFERENCE,QOS,REM_ADDRESS)
BEGIN;
  IF  (GLOBAL_STATE NE ACTIVE), THEN
    REASON := Session Layer Terminating
    CALL T_DISC (REFERENCE,REASON)
  ELSE
    REASON := Too Many Active Connections
    IF  (ACTIVE_CONN LT MAX_CONN), THEN
      CALL GETBLOCK (CCBLEN)
      REASON := RETURN
      IF  (RETURN EQ 0), THEN
        CALL GETBLOCK (650)
        REASON := RETURN
        IF  (RETURN EQ 0), THEN
          CCB.TDATA := POINTER
          CCB.TLEN := 650
          CALL CHAIN (CCB,REQCHAIN)
          CCB.REF := REFERENCE
          CCB.TCEP := (new Value)
          CCBSTATE := STATIC
          CCB.TR_EX := T_EXP
          CALL T_CONN (RESPONSE,CCB.REF,CCB.TCEP,1,CCBTDATA,CCBTLEN)
        ELSE
          CALL T_DISC (REFERENCE,REASON)
      ELSE
        CALL T_DISC (REFERENCE,REASON)
      ENDIF
    ELSE
      CALL T_DISC (REFERENCE,REASON)
    ENDIF
  ENDIF
END of Procedure CONNIND
```

6.5.2.3 T-connect.confirm procedure

A T-connect.confirm signals that the Transport connection has been established: the CONNECT SPDU may now be sent on the established Transport connection as normal data. All information necessary for the encoding of the CONNECT SPDU is supposedly stored in the CCB by the CONNECT procedure (see also "3.1.6 CONNECT event" pag. 91).

```
PROCEDURE CONCONF (REFERENCE,QOS)
BEGIN;
  REASON := Unknown Transport Identifier
  CALL SEARCH (REQCHAIN,REFERENCE)
  IF  (RETURN EQ 0), THEN
    IF  (CCBSTATE EQ STA1B), THEN
      CALL SEND (CCE)
    ELSE
      REASON := Invalid Tcep
      CALL T_DISC (CCE.TCEP,CCB.REF,REASON)
    ENDIF
  ELSE
    CALL MESSAGE ('LMS',REASON)
  ENDIF
END of Procedure CONCONF
```

6.5.2.4 T-disc.indication procedure

A T-disc.indication signals that a Transport connection has been released. This may happen either because an ABORT SPDU had already been sent, or due to Transport internal reasons.

```
PROCEDURE DISCIND (REFERENCE,REASON)
BEGIN;
  REASON := Unknown Transport Identifier
  CALL SEARCH (CONCHAIN,REFERENCE)
  IF  (RETURN EQ 0), THEN
    REASON := Unknown Session User
    CALL SEARCH (UCBCHAIN,CCBUSER)
    IF  (RETURN EQ 0), THEN
      CALL DECHAIN (CCB,REQCHAIN)
      CALL LOCK (UCB)
      CALL CHAIN (CCB,CCBCHAIN(UCB))
      CALL UNLOCK (UCB)
      IF  (CCBSTATE NE (STA1 OR STA16)), THEN
        TEXT := Abort Indication (Provider generated)
        CALL MESSAGE (CCBUSER,TEXT)
        CCBSTATE := STA1
      ELSE
        CCBSTATE := STA1
      ENDIF
    ENDIF
  ELSE
    CALL MESSAGE ('LMS',REASON)
  ENDIF
END of Procedure DISCIND
```

6.5.2.5 T-data.indication procedure

A T-data.indication signals that a SPDU (or more, in case of concatenation) has arrived for a given connection. The SPDU is first decoded, then the SPM input routines are called (see also "6.4.2 INPUT machine" pag. 205).

```
PROCEDURE DATAIND (REFERENCE,DATA_POINTER,DATA_LENGTH)
BEGIN;
  REASON := Unknown Transport Identifier
  CALL SEARCH (CONCHAIN,REFERENCE)
  IF  (RETURN EQ 0), THEN
    CALL LOCK (CCB)
    CCBTDATA_AVAILABLE := CCBTDATA_AVAILABLE - DATA_LENGTH
    CCB.TDATA := DATA_POINTER
    CCB.TLEN := DATA_LENGTH
    DO WHILE (RETURN EQ 0)
      CALL DECCDE (CCB)
      IF  (RETURN EQ 0), THEN
        CALL IN_PROT (CCBINPDU,CCB)
      ENDIF
    ENDDO
    CALL UNLOCK (CCB)
  ELSE
    CALL MESSAGE ('LMS',REASON)
  ENDIF
END of Procedure DATAIND
```

6.5.2.6 Decode procedure

This procedure performs the decoding function following the rules described in "3.4.4 Concatenation" pag. 25 of "First Part."

No detailed description can be given here of this procedure, as it depends mostly on the implementation language; however, in "6.7 Not formally described parts" pag. 238, an indication of the way this procedure works is given. It is worth to say here that:

- The arrived TSDU is addressed by the CCB.TDATAIN field of the CCB.

- The arrived buffer is kept until the last SPDU (in case of concatenation) has been decoded. After that, the buffer is released, the pointer is cleared, and any successive call to the DECODE routine will set the RETURN variable to a value different from zero.

- The DECODE routine, for any legal PDU sets some CCB fields: the list of actions and fields is given in "6.7 Not formally described parts" pag. 238.

6.5.2.7 Enable-transmission procedure

An Enable Transmission is an OSIRIDE Transport Layer interface signal which is used to indicate to the Session layer that a TSDU has been sent. The data block, which is addressed in the Enable transmission event, is released and a new block (of different length) is allocated for an Enable reception signal to the Transport. The length of the new block is computed by using the maximum length of data buffer assigned to the Transport for that connection, and the actual length of data buffer which the Transport owns for that connection.

```
PROCEDURE ENBLTRX (REFERENCE,DATA_POINTER,DATA_LENGTH)
BEGIN;
  REASON := Unknown Transport Identifier
  CALL SEARCH (CONCHAIN,REFERENCE)
  IF  (RETURN EQ 0), THEN
    CALL DELETE (DATA_POINTER)
    NEWLENGTH := CCBTMAX.LENGTH - CCBTDATA_AVAILABLE
    CALL GETBLOCK (NEWLENGTH)
    IF  (RETURN EQ 0), THEN
      CALL ENBLRCV (CCB.REF,POINTER,NEWLENGTH)
      CCBTDATA_AVAILABLE := CCBTMAX.LENGTH
    ENDIF
  ELSE
    CALL MESSAGE ('LMS',REASON)
  ENDIF
END of Procedure ENBITRX
```

6.6 SESSION MANAGEMENT FUNCTIONS

6.6.1 State Tables

The same finite states machine which applies in the case of the Upper Layer Interface is used for the Session Management Functions. This state machine is described in Fig. 71 pag. 165.

6.6.2 Session management routines

The SMF is composed of a number of procedures, some of which constitute the "nucleus" of the Interface, others being only service procedures. These latter are, very often, machine dependent and, as such, are only described by words.

6.6.2.1 Command_Scanner

The Command Scanner procedure analyzes the request which has been passed to the SMF by the Scheduler.

```
PROCEDURE COMMAND_SCANNER (REQUEST CONTROL BLOCK)
BEGIN;
  RETURN := Invalid Operation;
  CASE (OPERATION) OF
    START   : CALL START (USER)
    STOP    : CALL STOP (USER)
    FLUSH   : CALL FLUSH (USER)
    STATUS  : CALL STATUS (USER,NAME,CONNECTION_ID)
  END_OF_CASE
  MESSAGE (USER,RETURN)
END of Procedure COMMAND_SCANNER
```

6.6.2.2 Start procedure

The Start procedure enables the Session Layer to start working, by setting its GLOBAL_STATE variable to the value ACTIVE.

```
PROCEDURE START (USER)
BEGIN;
  RETURN := Not Authorized
  IF  (USER EQ 'LMS'), THEN
    RETURN := 0
    GLOBAL_STATE := ACTIVE
  ENDIF
END of Procedure START
```

6.6.2.3 Stop procedure

The Stop procedure disables the Session layer. All active connections are terminated by means of an ABORT request. Session users are alerted. The GLOBAL_STATE variable is set to the value OFF.

```
PROCEDURE STOP (USER)
BEGIN;
  RETURN := Not Authorized
  IF  (USER EQ 'LMS'), THEN
    GLOBAL_STATE := OFF
    CALL SEARCH (UCBCHAIN,'NEXT')
    DO WHILE (RETURN EQ 0)
      CALL LOCK (UCB)
      CALL SEARCH (CCBCHAIN,'NEXT')
      DO WHILE (RETURN EQ 0)
        CALL ABORT (UCENAME,CONNECTION_ID,'STOP')
        CALL SEARCH (CCBCHAIN,'NEXT')
      ENDDO
      RETURN := 0
      CALL DECHAIN (UCB,UCBCHAIN)
      CALL MESSAGE (UCBUSER,'STOP')
      CALL DELETE (UCB)
      USERS_NUMBER := USERS_NUMBER - 1
      CALL SEARCH (UCBCHAIN,'NEXT')
    ENDDO
    RETURN := 0
  ENDIF
END of Procedure STOP
```

6.6.2.4 Flush procedure

The Flush procedure inhibits any further new connection to be established, by setting the GLOBAL_STATE variable to the value FLUSHING.

```
PROCEDURE FLUSH (USER)
BEGIN;
  RETURN := Not Authorized
  IF  (USER EQ 'LMS'), THEN
    RETURN := 0
    GLOBAL_STATE := FLUSHING
  ENDIF
END of Procedure FLUSH
```

6.6.2.5 Status procedure

The Status procedure gives back information either on the entire Session layer, or on a particular Session user.

```
PROCEDURE STATUS (USER,NAME,CONNECTION_ID)
BEGIN;
  RETURN := Not Authorized
  IF  (USER EQ 'LMS'), THEN
    IF  (NAME EQ 'OSISESSION')
      COUNT := 1
      DO WHILE (COUNT EQ (USERS_NUMBER + 1))
        CALL SEARCH (UCBCHAIN,'NEXT')
        USNAME(COUNT) := UCBNAME
        USCONN(COUNT) := UCBACTOUT + UCBACTIN
        COUNT := COUNT + 1
      ENDDO
      RETURN := USERS_NUMBER,USNAME,USCONN
    ELSE
      CALL SEARCH (UCBCHAIN,NAME)
      COUNT := 1
      CONNUMBER := UCBACTOUT + UCBACTIN
      DO WHILE (COUNT EQ (CONNUMBER + 1))
        CALL SEARCH (CCBCHAIN,'NEXT')
        USNODENAME(COUNT) := CCBSYSTEM
        USSENT(COUNT) := CCBSENT
        USRECV(COUNT) := CCBRECV
        COUNT := COUNT + 1
      ENDDO
      RETURN := USERS_NUMBER,USNODENAME,USSENT,USRECV
    END_IF
  ENDIF
END of Procedure STATUS
```

6.7 NOT FORMALLY DESCRIBED PARTS

This section describes those routines which cannot be formalized in the pseudo-language used for all the rest of the Session layer description.

The routines are described in terms of what they do, which input parameters and which output parameters are set or used.

6.7.1 ENQUEUE routine

The ENQUEUE procedure is invoked when a PREPARE (resynchronize) arrives in certain states where it can not be handled immediately. The event is then "stored" in a queue, and it is processed only afterwards. Although in this present protocol only the PREPARE (resynchronize) event is stored, the mechanism is general, and could

be used for other purposes, either in further versions of the protocol, or for implementation needs.

This is to say that, when the ENQUEUE procedure is invoked, a block is added to the CCBSTQUEUE, which contains the code of the PDU just arrived.

6.7.2 RECALL routine

The RECALL routine is invoked when the process which possibly caused an SPDU to be enqueued is terminated. The RECALL routine takes the PDUs enqueued in the CCBSTQUEUE and drives the appropriate input protocol routines. At the moment, only the PREPARE SPDU may be present in this queue.

Note: There may be more that one PDU enqueued. When the RECALL routine is called, it is intended that **all** those PDUs are processed, i.e. the RECALL routine terminates only when the CCBSTQUEUE is empty.

6.7.3 CLEARQ routine

The CLEARQ routine is invoked when the PDUs enqueued by means of an ENQUEUE function have to be ignored, and the queue has to be cleared. After this routine, the CCBSTQUEUE is empty.

6.7.4 T_CONN routine

This routine is a part of the Session/Transport interface, and can not be precisely defined: it, however, requests the establishment of a Transport connection for a given CCB. The parameters necessary for issueing a T-connect are:

- Tcep-id (stored in the CCB.TCEP field)
- Remote System address (stored in the CCBSYSTEM field)
- TSAP Address (forced to 1 in OSIRIDE Release 1)
- Throughput (as specified in the Profile for that connection)
- Resilience (as specified in the Profile for that connection)

Note: No Reference has to be provided, because not yet assigned by the TRansport layer at connection establishment time.

6.7.5 T_DISC routine

This routine is a part of the Session/Transport interface, and can not be precisely defined: it, however, requests a Transport disconnection for a given CCB. The parameters necessary for issueing a T-disconnect are:

- Reference (stored in the CCB.REF field)
- Tcep-id (stored in the CCB.TCEP field)
- Reason (stored in the CBBREASON field)

6.7.6 T_DATA routine

This routine is a part of the Session/Transport interface, and can not be precisely defined: it, however, requests a Transport data transfer for a given CCB. The parameters necessary for issueing a T-data are:

- Reference (stored in the CCB.REF field)
- Tcep-id (stored in the CCB.TCEP field)
- Data (buffer address stored in the CBB.TRANS field)
- Data length (stored in the TRANSLEN field)

6.7.7 T_EXDATA routine

This routine is a part of the Session/Transport interface, and can not be precisely defined: it, however, requests a Transport expedited data transfer for a given CCB. The parameters necessary for issueing a T-ex-data are:

- Reference (stored in the CCB.REF field)
- Tcep-id (stored in the CCB.TCEP field)
- Data (buffer address stored in the CBB.TRANS field)
- Data length (stored in the TRANSLEN field)

6.7.8 ENBLRCV_routine

This routine is a part of the Session/Transport interface, and can not be precisely defined: it, however, is the way the Session layer can pass a new buffer to the Transport layer. The parameters necessary for issueing an Enable Reception are:

- Reference (stored in the CCB.REF field)
- Data buffer address
- Buffer length

Note: Neither the buffer address nor the buffer length are stored in any CCB field.

6.7.9 ENCODE_routine

The ENCODE procedure is invoked by the SEND routine to encode a PDU to be transmitted.

What follows is the list of all OSIRIDE legal SPDUs: for each SPDU, the list of CCB fields used to set the related parameters is given.

6.7.9.1 CONNECT

Parameters related to the CONNECT SPDU.

Calling SS-user Reference The CCB.USER field is used.

Common Reference The CCB.COMMON field is used.

Additional Reference Information The CCB.ADDREF field is used.

Protocol Options Not present in any CCB field. Set **always** to one.

TSDU Max size Not present in any CCB field. Set **always** to zero.

Version Number Not present in any CCB field. Set **always** to zero.

Initial Serial Number The value stored as a binary number in CCB.IN_NUMBER is used. It is set only if at least one of CCB.FU_MI or CCB.FU_MA or CCB.FU_ES is set.

Token Setting Item The following CCB fields are used:

CCB.DATA

CCB.MARK

CCB.SYNC

CCB.TERM

Session User requirements. The following CCB fields are used:

CCB.FU_HD

CCB.FU_DX

CCB.FU_ER

CCB.FU_TY

CCB.FU_TR

CCB.FU_MI

CCB.FU_MA

CCB.FU_RS

CCB.FU_EX

CCB.FU_AC

CCB.FU_CA

Calling SSAP identifier The CCBUSER field is used.

Called SSAP identifier The CCB.REMOTE_USER field is used.

User Data The field CCBINDATA is used as a pointer to the data buffer, and the field CCBINLEN as the length of the buffer.

6.7.9.2 ACCEPT

Parameters related to the ACCEPT SPDU.

Calling SS-user Reference The CCB.REMOTE_USER field is used.

Common Reference The CCB.COMMON field is used.

Additional Reference Information The CCB.ADDREF field is used.

Protocol Options Not present in any CCB field. Set always to one.

TSDU Max size Not present in any CCB field. Set always to zero.

Version Number Not present in any CCB field. Set always to zero.

Initial Serial Number The value stored as a binary number in CCB.IN_NUMBER. It is set only if at least one of CCB.FU_MI or CCB.FU_MA or CCB.FU_RS is set.

Token Setting Item The following CCB fields are used:

CCB.DATA

CCB.MARK

CCB.SYNC

CCB.TERM

Session User requirements. The following CCB fields are used:

CCB.FU_HD

CCB.FU_DX

CCB.FU_ER

CCB.FU_TY

CCB.FU_TR

CCB.FU_MI

CCB.FU_MA

CCB.FU_RS

CCB.FU_EX

CCB.FU_AC

CCB.FU_CA

Calling SSAP identifier The CCB.REMOTE_USER field is used.

Called SSAP identifier The CCBUSER field is used.

User Data The field CCB.TDATA is used as a pointer to the data buffer, and the field CCB.TLEN is used to set the length of the buffer.

6.7.9.3 REFUSE

Parameters related to the REFUSE SPDU.

Calling SS-user Reference The CCB.REMOTE_USER field is used.

Common Reference The CCB.COMMON field is used.

Additional Reference Information The CCB.ADDREF field is used.

Transport Disconnect Not present in any CCB field.

Session User requirements. The following CCB fields are used:

CCB.FU_HD

CCB.FU_DX

CCB.FU_ER

CCB.FU_TY

CCB.FU_TR

CCB.FU_MI

CCB.FU_MA

CCB.FU_RS

CCB.FU_EX

CCB.FU_AC

CCB.FU_CA

Version Number Not present in any CCB field. Set always to zero.

Reason Code The field CCB.TDATA is used as a pointer to the data buffer, and the field CCB.TLEN is set to the length of the buffer.

6.7.9.4 FINISH

Parameters related to the FINISH SPDU.

Transport Disconnect Not present in any CCB field.

User Data The field CCB.TDATA is used as a pointer to the data buffer, and the field CCB.TLEN is set to the buffer length.

6.7.9.5 DISCONNECT

Parameters related to the DISCONNECT SPDU.

User Data The field CCB.TDATA is used as a pointer to the data buffer, and the field CCB.TLEN is set to the buffer length.

6.7.9.6 NOT_FINISHED

Parameters related to the NOT FINISHED SPDU.

User Data — The field CCB.TDATA is used as a pointer to the data buffer, and the field CCB.TLEN is set to the buffer length.

6.7.9.7 ABORT

Parameters related to the ABORT SPDU.

Transport Disconnect — Not present in any CCB field and not set.

Reflect parameter values — Not present in any CCB field and not set.

User Data — The field CCB.TDATA is used as a pointer to the data buffer, and the field CCB.TLEN is set to the buffer length.

6.7.9.8 DATA_TRANSFER

Parameters related to the DATA TRANSFER SPDU.

Enclosure Item — Not present in any CCB field and not set.

User Information field — The field CCB.TDATA is used as a pointer to the input data buffer, and the field CCB.TLEN is used to indicate the length.

6.7.9.9 TYPED_DATA

Parameters related to the TYPED DATA SPDU.

Enclosure Item — Not present in any CCB field and not set.

User Information field — The field CCB.TDATA is used as a pointer to the input data buffer, and the field CCB.TLEN is used to indicate the length.

6.7.9.10 GIVE_TOKENS

Parameters related to the GIVE TOKENS SPDU.

Token Item — The following CCB fields are set 'TRUE' if the corresponding bit in the Token Item is to be given:

CCB.TOK_DT for the data token

CCB.TOK_MK for the minor sync token

CCB.TOK_SY for the major sync token

CCB.TOK_TR for the terminate token

6.7.9.11 PLEASE TOKENS

Parameters related to the PLEASE TOKENS SPDU.

Token Item The following CCB fields are set 'TRUE' if the corresponding bits in the Token Setting Item is to be asked for:

CCB.TOK_DT for the data token

CCB.TOK_MK for the minor sync token

CCB.TOK_SY for the major sync token

CCB.TOK_TR for the terminate token

User Data The field CCB.TDATA is used as a pointer to the input data buffer, and the field CCB.TLEN is used to indicate the length.

6.7.9.12 MINOR SYNC POINT

Parameters related to the MINOR SYNC POINT SPDU:

Sync Type Item If the field CCB_EXPLICIT is set to 'TRUE' this parameter is not set.

Serial Number The field CCBPOINT is used to set the specified value.

User Data The field CCB.TDATA is used as a pointer to the input data buffer, and the field CCB.TLEN is used to indicate the length.

6.7.9.13 MINOR SYNC ACK

Parameters related to the MINOR SYNC ACK SPDU:

Serial Number The field CCBPOINT is used to set the specified value.

User Data The field CCB.TDATA is used as a pointer to the input data buffer, and the field CCB.TLEN is used to indicate the length.

6.7.9.14 MAJOR SYNC POINT

Parameters related to the MAJOR SYNC POINT SPDU:

Sync Type Item This field is ignored in OSIRIDE, and no CCB fields are set by it.

Serial Number The field CCBPOINT is used to set the specified value.

User Data The field CCB.TDATA is used as a pointer to the input data buffer, and the field CCB.TLEN is used to indicate the length.

6.7.9.15 MAJOR SYNC ACK

Parameters related to the MAJOR SYNC ACK SPDU:

Serial Number The field CCBPOINT is used to set the specified value.

User Data The field CCB.TDATA is used as a pointer to the input data buffer, and the field CCB.TLEN is used to indicate the length.

6.7.9.16 RESYNCHRONIZE

Parameters related to the RESYNCHRONIZE SPDU:

Token Setting Item The following CCB fields are set 'TRUE' if the corresponding Token Setting Item bit is to be specified:

CCB.RES_DK for the data token

CCB.RES_MK for the minor sync token

CCB.RES_SK for the major sync token

CCB.RES_TK for the terminate token

Resync Type The field CCB.RESTYPE is used to set the specified value.

Serial Number The field CCBPOINT is used to set the specified value.

User Data The field CCB.TDATA is used as a pointer to the input data buffer, and the field CCB.TLEN is used to indicate the length.

6.7.9.17 RESYNCHRONIZE_ACK

Parameters related to the RESYNCHRONIZE ACK SPDU:

Token Setting Item The following CCB fields are set 'TRUE' if the corresponding Token Setting Item bit is to be specified:

CCB.RES_DK for the data token

CCB.RES_MK for the minor sync token

CCB.RES_SK for the major sync token

CCB.RES_TK for the terminate token

Serial Number The field CCBPOINT is used to set the specified value.

User Data The field CCB.TDATA is used as a pointer to the input data buffer, and the field CCB.TLEN is used to indicate the length.

6.7.9.18 PREPARE

Parameters related to the PREPARE SPDU:

Prepare Type The field CCBTYPE is used to set the specified PREPARE type.

6.7.10 DECODE_routine

The DECODE procedure is invoked by the LLI when a TSDU or EX-TSDU arrives. Its task is to decode the SPDU(s) contained in the TSDU, and to set appropriate parameters in the CCB, according to the arrived SPDU. When the last PDU has been processed, the incoming buffer is released.

What follows is the list of all OSIRIDE legal SPDUs: for each SPDU, the list of CCB fields set by the related parameters is given.

6.7.10.1 CONNECT

Parameters related to the CONNECT SPDU.

Calling SS-user Reference Stored in the CCB.REMOTE_USER field.

Common Reference Stored in the CCB.COMMON field.

Additional Reference Information Stored in the CCB.ADDREF field.

Protocol Options Not present in any CCB field. Set always to one.

TSDU Max size Not present in any CCB field. Set always to zero.

Version Number Not present in any CCB field. Set always to zero.

Initial Serial Number Stored as a binary number in CCB.IN_NUMBER. To be set only if at least one of CCB.FU_MI or CCB.FU_MA or CCB.FU_RS is set.

Token Setting Item The token setting item is stored in the following CCB fields:

CCB.DATA

CCB.MARK

CCB.SYNC

CCB.TERM

Session User requirements. The Session User requirements is stored in the following CCB fields:

CCB.FU_HD

CCB.FU_DX

CCB.FU_ER

CCB.FU_TY

CCB.FU_TR

CCB.FU_MI

CCB.FU_MA

CCB.FU_RS

CCB.FU_EX

CCB.FU_AC

CCB.FU_CA

Calling SSAP identifier The CCB.REMOTE_USER field is used.

Called SSAP identifier The CCBUSER field is used.

User Data The field CCBINDATA is used as a pointer to the data buffer, and the field CCBINLEN as the length of the buffer.

6.7.10.2 ACCEPT

Parameters related to the ACCEPT SPDU.

Calling SS-user Reference Not used. Should be equal to CCB.USER.

Common Reference Stored in the CCB.COMMON field.

Additional Reference Information Stored in the CCB.ADDREF field.

Protocol Options Not present in any CCB field. Set **always** to one.

TSDU Max size Not present in any CCB field. Set **always** to zero.

Version Number Not present in any CCB field. Set **always** to zero.

Initial Serial Number Stored as a binary number in CCB.IN_NUMBER. To be set only if at least one of CCB.FU_MI or CCB.FU_MA or CCB.FU_RS is set.

Token Setting Item The token setting item is stored in the following CCB fields:

CCB.DATA

CCB.MARK

CCB.SYNC

CCB.TERM

Session User requirements. The Session User requirements is stored in the following CCB fields:

CCB.FU_HD

CCB.FU_DX

CCB.FU_ER

CCB.FU_TY

CCB.FU_TR

CCB.FU_MI

CCB.FU_MA

CCB.FU_RS

CCB.FU_EX

CCB.FU_AC

CCB.FU_CA

Calling SSAP identifier The CCB.REMOTE_USER field is used.

Called SSAP identifier The CCBUSER field is used.

User Data The field CCBINDATA is used as a pointer to the data buffer, and the field CCBINLEN is used to set the length of the buffer.

6.7.10.3 REFUSE

Parameters related to the REFUSE SPDU.

Calling SS-user Reference Not used. Should be equal to CCB.USER.

Common Reference Stored in the CCB.COMMON field.

Additional Reference Information Stored in the CCB.ADDREF field.

Transport Disconnect Not present in any CCB field.

Session User requirements. The Session User requirements is stored in the following CCB fields:

CCB.FU_HD

CCB.FU_DX

CCB.FU_ER

CCB.FU_TY

CCB.FU_TR

CCB.FU_MI

CCB.FU_MA

CCB.FU_RS

CCB.FU_EX

CCB.FU_AC

CCB.FU_CA

Version Number Not present in any CCB field. Set always to zero.

Reason Code The field CCBTDATA is used as a pointer to the data buffer, and the field CCBINLEN is set to the length of the buffer.

6.7.10.4 FINISH

Parameters related to the FINISH SPDU.

Transport Disconnect Not present in any CCB field.

User Data The field CCBINDATA is used as a pointer to the data buffer, and the field CCBINLEN is set to the buffer length.

6.7.10.5 DISCONNECT

Parameters related to the DISCONNECT SPDU.

User Data The field CCBINDATA is used as a pointer to the data buffer, and the field CCBINLEN is set to the buffer length.

6.7.10.6 NOT FINISHED

Parameters related to the NOT FINISHED SPDU.

User Data The field CCBINDATA is used as a pointer to the data buffer, and the field CCBINLEN is set to the buffer length.

6.7.10.7 ABORT

Parameters related to the ABORT SPDU.

Transport Disconnect Not present in any CCB field and not set.

Reflect parameter values Not present in any CCB field and not set.

User Data The field CCBINDATA is used as a pointer to the data buffer, and the field CCBINLEN is set to the buffer length.

6.7.10.8 DATA TRANSFER

Parameters related to the DATA TRANSFER SPDU.

Enclosure Item Not present in any CCB field and not set.

User Information field The field NEWDATA is used as a pointer to the input data buffer, and the field NEW-LENGTH is used to indicate the length.

6.7.10.9 TYPED_DATA

Parameters related to the TYPED DATA SPDU.

Enclosure Item Not present in any CCB field and not set.

User Information field The field NEWDATA is used as a pointer to the input data buffer, and the field NEW-LENGTH is used to indicate the length.

6.7.10.10 GIVE TOKENS

Parameters related to the GIVE TOKENS SPDU.

Token Item The following CCB fields are set 'TRUE' if the corresponding token is given:

CCB.IN_DT for the data token

CCB.IN_MK for the minor sync token

CCB.IN_SY for the major sync token

CCB.IN_TR for the terminate token

6.7.10.11 PLEASE_TOKENS

Parameters related to the PLEASE TOKENS SPDU.

Token Item The following CCB fields are set 'TRUE' if the corresponding token is asked for:

CCB.IN_DT for the data token

CCB.IN_MK for the minor sync token

CCB.IN_SY for the major sync token

CCB.IN_TR for the terminate token

User Data The field CCBINDATA is used as a pointer to the input data buffer, and the field CCBINLEN is used to indicate the length.

6.7.10.12 MINOR_SYNC_POINT

Parameters related to the MINOR SYNC POINT SPDU:

Sync Type Item The field CCB_EXPLICIT is set to 'TRUE' if this parameter has the value 0 or is not present.

Serial Number The field CCBPOINT is set to the specified value.

User Data The field CCBINDATA is used as a pointer to the input data buffer, and the field CCBINLEN is used to indicate the length.

6.7.10.13 MINOR_SYNC_ACK

Parameters related to the MINOR SYNC ACK SPDU:

Serial Number The field CCBPOINT is set to the specified value.

User Data The field CCBINDATA is used as a pointer to the input data buffer, and the field CCBINLEN is used to indicate the length.

6.7.10.14 MAJOR_SYNC_POINT

Parameters related to the MAJOR SYNC POINT SPDU:

Sync Type Item This field is ignored in OSIRIDE, and no CCB fields are set by it.

Serial Number The field CCBPOINT is set to the specified value.

User Data The field CCBINDATA is used as a pointer to the input data buffer, and the field CCBINLEN is used to indicate the length.

6.7.10.15 MAJOR_SYNC_ACK

Parameters related to the MAJOR SYNC ACK SPDU:

Serial Number The field CCBPOINT is set to the specified value.

User Data The field CCBINDATA is used as a pointer to the input data buffer, and the field CCBINLEN is used to indicate the length.

6.7.10.16 RESYNCHRONIZE

Parameters related to the RESYNCHRONIZE SPDU:

Token Setting Item The following CCB fields are set 'TRUE' if the corresponding token is specified:

CCB.IN_DT for the data token

CCB.IN_MK for the minor sync token

CCB.IN_SY for the major sync token

CCB.IN_TR for the terminate token

Resync Type The field CCB.RESTYPE is set to the specified value.

Serial Number The field CCBPOINT is set to the specified value.

User Data The field CCBINDATA is used as a pointer to the input data buffer, and the field CCBINLEN is used to indicate the length.

6.7.10.17 RESYNCHRONIZE ACK

Parameters related to the RESYNCHRONIZE ACK SPDU:

Token Setting Item The following CCB fields are set 'TRUE' if the corresponding token is specified:

CCB.IN_DT for the data token

CCB.IN_MK for the minor sync token

CCB.IN_SY for the major sync token

CCB.IN_TR for the terminate token

Serial Number The field CCBPOINT is set to the specified value.

User Data The field CCBINDATA is used as a pointer to the input data buffer, and the field CCBINLEN is used to indicate the length.

6.7.10.18 PREPARE

Parameters related to the PREPARE SPDU:

Prepare Type The field CCBTYPE is set to the specified PREPARE type.

REFERENCES

[1] F. Caneschi: "The OSIRIDE Session Layer", Doc. OSIRIDE/83/SES/01, Pisa, April 1983.

[2] F. Caneschi, E. Gregori: "Guidelines for the implementation specifications of the OSIRIDE layers" Internal Report C83-5, Copyright CNUCE Pisa 1983.

[3] Information Processing Systems - Open Systems Interconnection - ISO Draft International Standard DIS 8326 Session Services Description.

[4] Information Processing Systems - Open Systems Interconnection - ISO Draft International Standard DIS 8327 Session Protocol Specification.

[5] Information Processing Systems - Open Systems Interconnection - ISO Draft International Standard 7498 - Basic Reference Model.

[6] F. Caneschi, E. Zucchelli: "The OSIRIDE File Transfer Service" Doc OSIRIDE/83/FTP/04 Pisa May 1983.

[7] ECMA-85 Standard Virtual File Protocol.

[8] F. Caneschi, E. Gregori, C. Menchi: "ISIDE, or the OSIRIDE Access Method" Doc OSIRIDE/83/FDT/02, Pisa April 1083.

[9] Information Processing Systems - Open Systems Interconnection - ISO Draft Proposal DP 8072 Transport Layer Services Description.

[10] E. Gregori: "The OSIRIDE Transport Layer" Doc OSIRIDE/83/TRA/01, Pisa June 1083.

Heading ID's

id	File	Pag.	Heading Ref.
first	OSISES02	xii	First Part 77, 81, 113, 118, 151, 204, 205, 234
def	SES1	2	1.2 Definitions
sserv	SES1	7	2.0 Session Service 205
hdpx	SES1	13	2.3 Half Duplex Functional Unit 15, 16
excon	SES1	25	3.4.4 Concatenation 38, 51, 113, 118, 204, 234
sconn	SES1	29	4.1 CONNECT SPDU 22
sacc	SES1	31	4.2 ACCEPT SPDU 22
sref	SES1	33	4.3 REFUSE SPDU 22
sfin	SES1	34	4.4 FINISH SPDU 22
sdis	SES1	35	4.5 DISCONNECT SPDU 22
sabo	SES1	36	4.6 ABORT SPDU 22
saba	SES1	37	4.7 ABORT ACCEPT SPDU 22
snot	SES1	38	4.8 NOT FINISHED SPDU 22
dspdu	SES1	38	4.9 DATA TRANSFER SPDU 22, 41
spto	SES1	40	4.11 PLEASE TOKENS SPDU 22, 22, 22, 22
styp	SES1	41	4.12 TYPED DATA SPDU 22
smin	SES1	42	4.13 MINOR SYNCHRONIZATION POINT SPDU 22
smic	SES1	43	4.14 MINOR SYNC ACK SPDU 22
smaj	SES1	43	4.15 MAJOR SYNCHRONIZATION POINT SPDU 22
smac	SES1	44	4.16 MAJOR SYNC ACK SPDU 22
sres	SES1	45	4.17 RESYNCHRONIZE SPDU 22
rcont	SES1	47	4.17.4 Resynchronization contention 46, 47
srak	SES1	48	4.18 RESYNCHRONIZE ACK SPDU 22
spre	SES1	49	4.19 PREPARE SPDU 22
enc	SES1	51	5.0 Structure and encoding of SPDUs 151
struh	SES1	51	5.2 SPDU structure
pi	SES1	52	5.2.3 PI units 52
tables	SES1	55	5.3 SPDU identifiers and associated parameter fields

Figure ID's

id	File	Pag.	Figure Ref.
con	SES1	8	1: 8
dat	SES1	11	2: 11
rel	SES1	11	3: 11
abo	SES1	12	4: 12
abp	SES1	12	5: 12
typ	SES1	13	6: 13
futk	SES1	14	7: 13, 15, 16
tok	SES1	14	8: 14
top	SES1	15	9: 14
min	SES1	16	10: 15
maj	SES1	17	11: 16
res	SES1	17	12: 17
colre	SES1	19	13: 18
tran	SES1	20	14: 20
tses	SES1	22	15: 21
pconc	SES1	26	16: 25
valcon	SES1	26	17: 25, 68, 68
valor	SES1	26	18: 25, 68, 68
tsdu	SES1	27	19: 25
cont	SES1	47	20: 47
pretab	SES1	50	21: 49
stru2	SES1	53	22: 53
econ	SES1	56	23:
eacc	SES1	60	24:
eref	SES1	62	25:
efin	SES1	64	26:
edis	SES1	65	27:
enfi	SES1	65	28:
eabo	SES1	66	29:
eple	SES1	67	30:
egiv	SES1	68	31:
epre	SES1	69	32:
e2td	SES1	69	33:
e2dt	SES1	70	34:
e2mi	SES1	71	35:
e2mic	SES1	72	36:
e2ma	SES1	72	37:
e2mac	SES1	73	38:
e2res	SES1	74	39:
e2rak	SES1	75	40:
stru	SES2	80	41: 79, 79
pstate	SESTAB	119	42: 118, 118
pevn	SESTAB	120	43: 118, 118
pevn1	SESTAB	121	44: 118
pevn2	SESTAB	122	45: 118
pacro	SESTAB	123	46: 118
act	SESTAB	130	47: 129
act1	SESTAB	131	48: 129
prd1	SESTAB	132	49:
prd2	SESTAB	133	50:
prd3	SESTAB	134	51:
prd4	SESTAB	135	52:
indx	SESTAB	136	53: 136
pcon	SESTAB	137	54: 129, 136
pda2	SESTAB	138	55: 136

pda5	SESTAB	138	56:	136
psyn	SESTAB	139	57:	136
psy1	SESTAB	139	58:	136
pmaj	SESTAB	140	59:	136
pma1	SESTAB	141	60:	136
pres	SESTAB	141	61:	136
presp	SESTAB	142	62:	136
pre1	SESTAB	143	63:	136
pre1p	SESTAB	144	64:	136
pdsc	SESTAB	145	65:	136
pds1	SESTAB	146	66:	136
pabo	SESTAB	147	67:	136
pabo1	SESTAB	147	68:	136
ptok	SESTAB	148	69:	136
pto1	SESTAB	148	70:	129, 136
ins	SES2	163	71:	
swa	SES2	164	71:	
ulista	SES2	165	71:	164, 229, 235
scann	SES2	166	72:	
op	SES2	167	72:	
clo	SES2	167	72:	
rea	SES2	168	72:	
rec	SES2	169	72:	
res2	SES2	170	72:	
rsc	SES2	170	72:	
rsa	SES2	170	72:	
con2	SES2	171	72:	
rep	SES2	172	72:	
cor	SES2	173	72:	
rsy	SES2	173	72:	
rsp	SES2	174	72:	
rsg	SES2	174	72:	
pre	SES2	175	72:	
pfi	SES2	176	72:	
tra	SES2	177	72:	
dat2	SES2	178	72:	
dat4	SES2	178	72:	
dat3	SES2	179	72:	
dat5	SES2	179	72:	
tos	SES2	180	72:	
syn	SES2	180	72:	
rre	SES2	180	72:	
abo2	SES2	181	72:	
fin	SES2	181	72:	
gtb	SES2	182	72:	
gde	SES2	182	72:	
cha	SES2	183	72:	
dec	SES2	184	72:	
sea	SES2	185	72:	
loo	SES2	186	72:	
acin	STAPROG	207	72:	
proter	STAPROG	208	72:	
cnin	STAPROG	208	72:	
dtin	STAPROG	210	72:	
dnin	STAPROG	211	72:	
fnin	STAPROG	212	72:	
gtin	STAPROG	214	72:	
maain	STAPROG	215	72:	
mapin	STAPROG	216	72:	
miain	STAPROG	217	72:	
mipin	STAPROG	218	72:	
nfin	STAPROG	219	72:	

ptin	STAPROG	220	72:	
prin	STAPROG	221	72:	
prmcd	STAPROG	221	72:	
prra	STAPROG	222	72:	
prrs	STAPROG	222	72:	
rain	STAPROG	223	72:	
rsin	STAPROG	224	72:	
rsact	STAPROG	226	72:	
llista	SES2	230	72:	229
llirou	SES2	231	73:	
tconi	SES2	232	73:	
tcond	SES2	233	73:	
tdisc	SES2	233	73:	
tdata	SES2	234	73:	
tenab	SES2	235	73:	
mscan	SES2	236	73:	
msta	SES2	236	73:	
msto	SES2	237	73:	
mflu	SES2	237	73:	
mtat	SES2	238	73:	

Footnote ID's

id	File	Pag.	Footnote Ref.	
fuo	SES1	7	1:	7
def	SES1	9	2:	9
pgi	SES1	52	3:	51
li	SES1	52	4:	52
local	SES2	111	5:	111
tcep	SES2	111	6:	111
abt	MARINA1	153	7:	153

Imbed Trace