

INTERNATIONAL STANDARD

NORME INTERNATIONALE

**Industrial communication networks – Fieldbus specifications –
Part 6-7: Application layer protocol specification – Type 7 elements**

**Réseaux de communication industriels – Spécifications des bus de terrain –
Partie 6-7: Spécifications de protocole de la couche d'application – Éléments
de Type 7**

IECNORM.COM : Click to view the full PDF of IEC 61158-6-7:2007



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2007 IEC, Geneva, Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either IEC or IEC's member National Committee in the country of the requester.

If you have any questions about IEC copyright or have an enquiry about obtaining additional rights to this publication, please contact the address below or your local IEC member National Committee for further information.

Droits de reproduction réservés. Sauf indication contraire, aucune partie de cette publication ne peut être reproduite ni utilisée sous quelque forme que ce soit et par aucun procédé, électronique ou mécanique, y compris la photocopie et les microfilms, sans l'accord écrit de la CEI ou du Comité national de la CEI du pays du demandeur.

Si vous avez des questions sur le copyright de la CEI ou si vous désirez obtenir des droits supplémentaires sur cette publication, utilisez les coordonnées ci-après ou contactez le Comité national de la CEI de votre pays de résidence.

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

Useful links:

IEC publications search - www.iec.ch/searchpub

The advanced search enables you to find IEC publications by a variety of criteria (reference number, text, technical committee,...).

It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available on-line and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary (IEV) on-line.

Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de la CEI

La Commission Electrotechnique Internationale (CEI) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications CEI

Le contenu technique des publications de la CEI est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Liens utiles:

Recherche de publications CEI - www.iec.ch/searchpub

La recherche avancée vous permet de trouver des publications CEI en utilisant différents critères (numéro de référence, texte, comité d'études,...).

Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

Just Published CEI - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications de la CEI. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (VEI) en ligne.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

INTERNATIONAL STANDARD

NORME INTERNATIONALE

**Industrial communication networks – Fieldbus specifications –
Part 6-7: Application layer protocol specification – Type 7 elements**

**Réseaux de communication industriels – Spécifications des bus de terrain –
Partie 6-7: Spécifications de protocole de la couche d'application – Éléments
de Type 7**

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

COMMISSION
ELECTROTECHNIQUE
INTERNATIONALE

PRICE CODE
CODE PRIX

XG

ICS 25.040.40; 35.100.70

ISBN 978-2-8322-1022-2

**Warning! Make sure that you obtained this publication from an authorized distributor.
Attention! Veuillez vous assurer que vous avez obtenu cette publication via un distributeur agréé.**

CONTENTS

FOREWORD.....	6
INTRODUCTION.....	8
1 Scope.....	9
1.1 General.....	9
1.2 Specifications.....	9
1.3 Conformance.....	9
2 Normative references	10
3 Terms, definitions, symbols, abbreviations and conventions	10
3.1 Terms and definitions from other ISO/IEC standards	10
3.2 Terms and definitions from IEC 61158-5-7.....	11
3.3 Additional terms and definitions.....	12
3.4 Abbreviations and symbols.....	16
3.5 Conventions	16
3.6 Conventions used in state machines	16
4 Abstract syntax of data type	17
4.1 Data abstract syntax specification	17
4.2 FAL PDU abstract syntax	21
5 Transfer syntaxes.....	22
5.1 Compact encoding.....	22
5.2 Data type encoding	23
6 Structure of protocol machines	82
7 AP-context state machine.....	83
8 Sub-MMS FAL service protocol machine (FSPM).....	83
8.1 General.....	83
8.2 Projection of the SUB-MMS PDUs on the MCS services	83
8.3 Projection of the SUB-MMS abort service on the MCS services.....	83
8.4 Construction of a SUB-MMS-PDU from a service primitive	84
8.5 Extraction of a valid service primitive from a SUB-MMS-PDU	84
8.6 Negotiation of an abstract syntax and a transfer syntax commonly called presentation-context.....	84
8.7 Identification of the SUB-MMS core abstract syntax	86
8.8 Identification of the application context name	87
8.9 Identification of the ASE of the core abstract syntax and the transfer syntax	87
9 Association relationship protocol machine (ARPM)	87
10 DLL mapping protocol machine (DMPM).....	88
10.1 MPS ARPM and DMPM	88
10.2 MCS ARPM and DMPM.....	99
11 Protocol options	135
11.1 Conformances classes	135
Bibliography.....	155
Figure 1 – Example of an evaluation net	17
Figure 2 – Encoding of a CompactValue	22
Figure 3 – Organisation of the bits and octets within a PDU	24

Figure 4 – Encoding of a Bitstring	27
Figure 5 – Encoding of a Floating point value	28
Figure 6 – Encoding of a structure	29
Figure 7 – Encoding of a Boolean array	30
Figure 8 – Representation of a MCS PDU	36
Figure 9 – Relationships among Protocol Machines and Adjacent Layers	82
Figure 10 – A_Readloc service evaluation net	88
Figure 11 – A_Writeloc service evaluation net.....	89
Figure 12 – A_Update service evaluation net	90
Figure 13 – A_Readfar service evaluation net.....	91
Figure 14 – A_Writefar service evaluation net.....	93
Figure 15 – A_Sent service evaluation net	94
Figure 16 – A_Received service evaluation net.....	94
Figure 17 – Association establishment: Requester element state machine	101
Figure 18 – Association establishment: Responder element state machine	102
Figure 19 – Association termination: Requester element state machine	104
Figure 20 – Association termination: Responder element state machine	106
Figure 21 – Association revocation: Requester element state machine	107
Figure 22 – Association revocation: Acceptor element state machine.....	108
Figure 23 – Interactions between state machine in an associated mode data transfer	110
Figure 24 – Transfer service: Requester element state machine	114
Figure 25 – Transfer service: Acceptor element state machine	115
Figure 26 – Unacknowledged transfer: Requester element state machine	116
Figure 27 – Unacknowledged transfer: Acceptor element state machine	116
Figure 28 – Acknowledged transfer: Requester element state machine	118
Figure 29 – Acknowledged transfer: Acceptor element state machine	119
Figure 30 – Numbering mechanism state machine	120
Figure 31 – Retry mechanism state machine.....	122
Figure 32 – Anticipation mechanism state machine	125
Figure 33 – Segmentation mechanism state machine.....	127
Figure 34 – Reassembly mechanism state machine	129
Figure 35 – Interaction of state machine in a non associated data transfer	131
Figure 36 – Unacknowledged transfer: Requester element state machine	132
Figure 37 – Unacknowledged transfer: Acceptor element state machine	132
Figure 38 – Acknowledged transfer: Requester element state machine	133
Figure 39 – Acknowledged transfer: Acceptor element state machine	134
Table 1 – Example of encoding of a SEQUENCE	19
Table 2 – Example of encoding of a SEQUENCE OF	19
Table 3 – Example of encoding of a CHOICE	20
Table 4 – Example of encoding of an object identifier	21
Table 5 – Example of encoding of a PDU	22
Table 6 – MPS PDU types	25

Table 7 – Fields of a CompactValuePDU	25
Table 8 – Fields of a VariableDescriptionPDU.....	32
Table 9 – Fields of an AccessDescriptionPDU	33
Table 10 – Fields of a TypeDescriptionPDU.....	34
Table 11 – Fields of a ListDescriptionPDU	35
Table 12 – Coding of the different MCS PDU types	37
Table 13 – Coding of the variable part of the PDU	37
Table 14 – Structure of association establishment request.....	38
Table 15 – Structure of an associated establishment response	42
Table 16 – Structure of an association termination request	44
Table 17 – Structure of an association termination response	44
Table 18 – Structure of an association revocation request	45
Table 19 – Structure of an associated transfer request	46
Table 20 – Structure of an associated transfer acknowledgement	46
Table 21 – Structure of a non-associated transfer request	47
Table 22 – Structure of a non-associated transfer acknowledgement	48
Table 23 – Definitions of object classes	50
Table 24 – Definition of Sub-MMS Services	51
Table 25 – Structure of the antiduplication list.....	123
Table 26 – Structure of the reassembly list	128
Table 27 – PV_R/W parameter values	136
Table 28 – PV_IND parameter values	136
Table 29 – PV_LIS parameter values	136
Table 30 – Constraints on PV_LIS parameter	137
Table 31 – PV_AT parameter values.....	137
Table 32 – PV_RE parameter values	137
Table 33 – PV_UT parameter values.....	137
Table 34 – Constraints on PV_RE parameter	137
Table 35 – PH_R_A parameter values	138
Table 36 – PH_R_S parameter values	138
Table 37 – PH_R_P parameter values	138
Table 38 – PH_P_A parameter values	139
Table 39 – PH_P_S parameter values	139
Table 40 – PH_P_P parameter values	139
Table 41 – PH_COH parameter values	139
Table 42 – PH_FIA parameter values.....	140
Table 43 – PH_SPF parameter values	140
Table 44 – PH_SPM parameter values.....	140
Table 45 – PH_ACC parameter values.....	141
Table 46 – PH_RES parameter values.....	141
Table 47 – PH_AK parameter values	141
Table 48 – PH_RA parameter values	141
Table 49 – PH_SR parameter values	141

Table 50 – PH_CF parameter values	142
Table 51 – Constraints on PH_RA parameter	142
Table 52 – Constraints on PH_SR parameter	142
Table 53 – PT_OCT parameter values	142
Table 54 – PT_BIN parameter values	143
Table 55 – PT_VIS parameter values	143
Table 56 – PT_BOO parameter values	143
Table 57 – PT_BCD parameter values	143
Table 58 – PT_BTM parameter values	144
Table 59 – PT_INT parameter values	144
Table 60 – PT_UNO parameter values	144
Table 61 – PT_FPT parameter values	144
Table 62 – PT_GTM parameter values	145
Table 63 – PT_TAB parameter values	145
Table 64 – PT_STR parameter values	145
Table 65 – Constraints on PT_TAB parameter	146
Table 66 – Constraints on PT_STR parameter	146
Table 67 – Conformance classes for environment management	149
Table 68 – Conformance classes for VMD management	150
Table 69 – Conformance classes for PI management	151
Table 70 – Conformance classes for domain management	152
Table 71 – Conformance classes for variable/variable list management	153
Table 72 – Conformance classes for event management	154

INTERNATIONAL ELECTROTECHNICAL COMMISSION

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-7: Application layer protocol specification – Type 7 elements

FOREWORD

- 1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
- 2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
- 3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
- 4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
- 5) IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
- 6) All users should ensure that they have the latest edition of this publication.
- 7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
- 8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
- 9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

NOTE Use of some of the associated protocol types is restricted by their intellectual-property-right holders. In all cases, the commitment to limited release of intellectual-property-rights made by the holders of those rights permits a particular data-link layer protocol type to be used with physical layer and application layer protocols in Type combinations as specified explicitly in the IEC 61784 series. Use of the various protocol types in other combinations may require permission from their respective intellectual-property-right holders.

International Standard IEC 61158-6-7 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

This first edition and its companion parts of the IEC 61158-6 subseries cancel and replace IEC 61158-6:2003. This edition of this part constitutes an editorial revision.

This edition of IEC 61158-6 includes the following significant changes from the previous edition:

- a) deletion of the former Type 6 fieldbus for lack of market relevance;
- b) addition of new types of fieldbuses;
- c) partition of part 6 of the third edition into multiple parts numbered -6-2, -6-3, ...

This bilingual version (2013-09) corresponds to the monolingual English version, published in 2007-12.

The text of this standard is based on the following documents:

FDIS	Report on voting
65C/476/FDIS	65C/487/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

The French version of this standard has not been voted upon. This publication has been drafted in accordance with ISO/IEC Directives, Part 2.

The committee has decided that the contents of this publication will remain unchanged until the maintenance result date indicated on the IEC web site under <http://webstore.iec.ch> in the data related to the specific publication. At this date, the publication will be:

- reconfirmed;
- withdrawn;
- replaced by a revised edition, or
- amended.

NOTE The revision of this standard will be synchronized with the other parts of the IEC 61158 series.

The list of all the parts of the IEC 61158 series, under the general title *Industrial communication networks – Fieldbus specifications*, can be found on the IEC web site.

INTRODUCTION

This part of IEC 61158 is one of a series produced to facilitate the interconnection of automation system components. It is related to other standards in the set as defined by the “three-layer” fieldbus reference model described in IEC/TR 61158-1.

The application protocol provides the application service by making use of the services available from the data-link or other immediately lower layer. The primary aim of this standard is to provide a set of rules for communication expressed in terms of the procedures to be carried out by peer application entities (AEs) at the time of communication. These rules for communication are intended to provide a sound basis for development in order to serve a variety of purposes:

- as a guide for implementors and designers;
- for use in the testing and procurement of equipment;
- as part of an agreement for the admittance of systems into the open systems environment;
- as a refinement to the understanding of time-critical communications within OSI.

This standard is concerned, in particular, with the communication and interworking of sensors, effectors and other automation devices. By using this standard together with other standards positioned within the OSI or fieldbus reference models, otherwise incompatible systems may work together in any combination.

IECNORM.COM : Click to view the full PDF of IEC 61158-6-7:2007

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-7: Application layer protocol specification – Type 7 elements

1 Scope

1.1 General

The fieldbus application layer (FAL) provides user programs with a means to access the fieldbus communication environment. In this respect, the FAL can be viewed as a “window between corresponding application programs.”

This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 7 fieldbus. The term “time-critical” is used to represent the presence of a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

This standard specifies interactions between remote applications and defines the externally visible behavior provided by the Type 7 fieldbus application layer in terms of

- a) the formal abstract syntax defining the application layer protocol data units conveyed between communicating application entities;
- b) the transfer syntax defining encoding rules that are applied to the application layer protocol data units;
- c) the application context state machine defining the application service behavior visible between communicating application entities;
- d) the application relationship state machines defining the communication behavior visible between communicating application entities.

The purpose of this standard is to define the protocol provided to

- define the wire-representation of the service primitives defined in IEC 61158-5-7, and
- define the externally visible behavior associated with their transfer.

This standard specifies the protocol of the Type 7 fieldbus application layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498) and the OSI application layer structure (ISO/IEC 9545).

1.2 Specifications

The principal objective of this standard is to specify the syntax and behavior of the application layer protocol that conveys the application layer services defined in IEC 61158-5-7.

A secondary objective is to provide migration paths from previously-existing industrial communications protocols. It is this latter objective which gives rise to the diversity of protocols standardized in parts of the IEC 61158-6 series.

1.3 Conformance

This standard does not specify individual implementations or products, nor does it constrain the implementations of application layer entities within industrial automation systems.

There is no conformance of equipment to the application layer service definition standard. Instead, conformance is achieved through implementation of this application layer protocol specification.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 60559, *Binary floating-point arithmetic for microprocessor systems*

IEC 61158-3-7, *Industrial communication networks – Fieldbus specifications – Part 3-7: Data-link layer service definition – Type 7 elements*

IEC 61158-4-7, *Industrial communication networks – Fieldbus specifications – Part 4-7: Data-link layer protocol specification – Type 7 elements*

IEC 61158-5-7, *Industrial communication networks – Fieldbus specifications – Part 5-7: Application layer service definition – Type 7 elements*

ISO/IEC 7498-1, *Information technology – Open Systems Interconnection – Basic Reference Model – Part 1: The Basic Model*

ISO/IEC 8824, *Information technology – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1)*

ISO/IEC 8825, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*

ISO/IEC 9506-2, *Industrial automation systems – Manufacturing Message Specification – Part 2: Protocol specification*

ISO/IEC 9545, *Information technology – Open Systems Interconnection – Application Layer structure*

3 Terms, definitions, symbols, abbreviations and conventions

For the purposes of this document, the following definitions apply.

3.1 Terms and definitions from other ISO/IEC standards

3.1.1 Terms and definitions from ISO/IEC 7498-1

- a) abstract syntax
- b) application entity
- c) application process
- d) application protocol data unit
- e) application service element
- f) application entity invocation
- g) application process invocation
- h) application transaction
- i) presentation context
- j) real open system
- k) transfer syntax

3.1.2 Terms and definitions from ISO/IEC 9545

- a) application-association
- b) application-context
- c) application context name
- d) application-entity-invocation
- e) application-entity-type
- f) application-process-invocation
- g) application-process-type
- h) application-service-element
- i) application control service element

3.1.3 Terms and definitions from ISO/IEC 8824

- a) object identifier
- b) type
- c) value
- d) simple type
- e) structured type
- f) component type
- g) tag
- h) Boolean type
- i) true
- j) false
- k) integer type
- l) bitstring type
- m) octetstring type
- n) null type
- o) sequence type
- p) sequence of type
- q) choice type
- r) tagged type
- s) any type
- t) module
- u) production

3.1.4 Terms and definitions from ISO/IEC 8825

- a) encoding (of a data value)
- b) data value
- c) identifier octets (the singular form is used in this standard)
- d) length octet(s) (both singular and plural forms are used in this standard)
- e) contents octets

3.2 Terms and definitions from IEC 61158-5-7

- a) application relationship
- b) conveyance path
- c) client
- d) dedicated AR
- e) dynamic AR
- f) error class
- g) error code
- h) name
- i) numeric identifier
- j) peer
- k) pre-defined AR endpoint
- l) pre-established AR endpoint
- m) publisher
- n) subscriber

3.3 Additional terms and definitions

3.3.1

allocate

take a resource from a common area and assign that resource for the exclusive use of a specific entity

3.3.2

application

function or data structure for which data is consumed or produced

3.3.3

application objects

multiple object classes that manage and provide a run time exchange of messages across the network and within the network device

3.3.4

attribute

description of an externally visible characteristic or feature of an object

NOTE The attributes of an object contain information about variable portions of an object. Typically, they provide status information or govern the operation of an object. Attributes may also affect the behaviour of an object. Attributes are divided into class attributes and instance attributes.

3.3.5

behaviour

indication of how an object responds to particular events

3.3.6

called

service user or a service provider that receives an indication primitive or a request APDU

3.3.7

calling

service user or a service provider that initiates a request primitive or a request APDU

3.3.8

class

set of objects, all of which represent the same kind of system component

NOTE A class is a generalisation of an object; a template for defining variables and methods. All objects in a class are identical in form and behaviour, but usually contain different data in their attributes.

3.3.9

class attributes

attribute that is shared by all objects within the same class

3.3.10

class code

unique identifier assigned to each object class

3.3.11

class specific service

service defined by a particular object class to perform a required function which is not performed by a common service

NOTE A class specific object is unique to the object class which defines it

3.3.12**client**

- a) object which uses the services of another (server) object to perform a task
- b) initiator of a message to which a server reacts

3.3.13**clock**

device providing a measurement of the passage of time since a defined epoch

NOTE There are two types of clocks in IEC 61588, boundary clocks and ordinary clocks.

3.3.14**communication objects**

components that manage and provide a run time exchange of messages across the network

EXAMPLES Connection Manager object, Unconnected Message Manager (UCMM) object, Message Router object.

3.3.15**connection**

logical binding between application objects that may be within the same or different devices

NOTE Connections may be either point-to-point or multipoint.

3.3.16**connection ID (CID)**

identifier assigned to a transmission that is associated with a particular connection between producers and consumers, providing a name for a specific piece of application information

3.3.17**connection point**

buffer which is represented as a subinstance of an Assembly object

3.3.18**consume**

act of receiving data from a producer

3.3.19**consumer**

node or sink that is receiving data from a producer

3.3.20**consuming application**

application that consumes data

3.3.21**cyclic**

repetitive in a regular manner

3.3.22**device**

physical hardware connected to the link

NOTE: A device may contain more than one node.

3.3.23**device profile**

collection of device dependent information and functionality providing consistency between similar devices of the same device type

3.3.24

end node

producing or consuming node

3.3.25

end point

one of the communicating entities involved in a connection

3.3.26

error

discrepancy between a computed, observed or measured value or condition and the specified or theoretically correct value or condition

3.3.27

frame

denigrated synonym for DLPDU

3.3.28

instance

the actual physical occurrence of an object within a class, identifying one of many objects within the same object class.

EXAMPLE California is an instance of the object class state.

NOTE The terms object, instance, and object instance are used to refer to a specific instance.

3.3.29

instance attribute

attribute that is unique to an object instance and not shared by the object class

3.3.30

instantiated

object that has been created in a device

3.3.31

interoperability

capability of User Layer entities to perform coordinated and cooperative operations using the services of the FAL

3.3.32

little endian

Describes a model of memory organisation which stores the least significant octet at the lowest address, or for transfer, which transfers the lowest order octet first

3.3.33

management information

network accessible information that supports the management of the Fieldbus environment

3.3.34

member

piece of an attribute that is structured as an element of an array

3.3.35

message router

object within a node that distributes messaging requests to appropriate application objects

3.3.36

multipoint connection

connection from one node to many

NOTE Multipoint connections allow messages from a single producer to be received by many consumer nodes.

3.3.37

network

a set of nodes connected by some type of communication medium, including any intervening repeaters, bridges, routers and lower-layer gateways

3.3.38

object

abstract representation of a particular component within a device, usually a collection of related data (in the form of variables) and methods (procedures) for operating on that data that have clearly defined interface and behaviour

3.3.39

object specific service

service unique to the object class which defines it

3.3.40

originator

client responsible for establishing a connection path to the target

3.3.41

point-to-point connection

connection that exists between exactly two application objects

3.3.42

produce

act of sending data to be received by a consumer

3.3.43

producer

node that is responsible for sending data

3.3.44

receiving

service user that receives a confirmed primitive or an unconfirmed primitive, or a service provider that receives a confirmed APDU or an unconfirmed APDU

3.3.45

resource

resource is a processing or information capability of a subsystem

3.3.46

sending

service user that sends a confirmed primitive or an unconfirmed primitive, or a service provider that sends a confirmed APDU or an unconfirmed APDU

3.3.47

service

operation or function than an object and/or object class performs upon request from another object and/or object class

3.4 Abbreviations and symbols

ASCII	American Standard Code for Information Interchange
CID	connection ID
DLL	data-link layer
PDU	protocol data unit
OSI	open systems interconnection (see ISO/IEC 7498)
Rcv	receive
Rx	receive
SDU	service data unit
STD	state transition diagram, used to describe object behaviour
TPDU	transport protocol data unit
Tx	transmit
Xmit	transmit

3.5 Conventions

3.5.1 General concept

The FAL is defined as a set of object-oriented ASEs. Each ASE is specified in a separate subclause. Each ASE specification is composed of three parts: its class definitions, its services, and its protocol specification. The first two are contained in IEC 61158-5. The protocol specification for each of the ASEs is defined in this standard.

The class definitions define the attributes of the classes supported by each ASE. The attributes are accessible from instances of the class using the Management ASE services specified in IEC 61158-5 standard. The service specification defines the services that are provided by the ASE.

This standard uses the descriptive conventions given in ISO/IEC 10731.

3.5.2 Conventions for class definitions

The Data Link Layer mapping definitions are described using templates. Each template consists of a list of attributes for the class. The general form of the template is defined in IEC 61158-5.

3.5.3 Abstract syntax conventions

When the "optionalParametersMap" parameter is used, a bit number which corresponds to each OPTIONAL or DEFAULT production is given as a comment.

3.6 Conventions used in state machines

3.6.1 General

This subclause recalls the main definitions associated with the three specification techniques used : 'Evaluation Networks', 'Abstract objects', 'Service Conventions'.

3.6.2 Evaluation networks

To be helpful for the reader some mechanisms are explained by a graphical representation as shown in Figure 1. These representations are based on evaluation network built on an oriented graph with three types of nodes:

- the states, represented by a disk,

- the requests represented by a rectangle,
- the transitions, represented by a horizontal line.

The evaluation network is built according to the following principle:

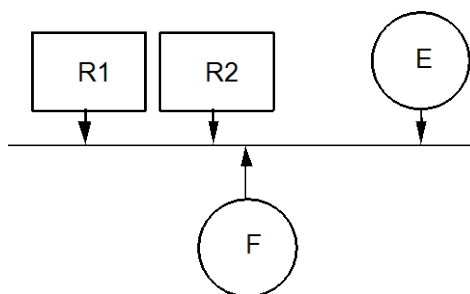


Figure 1 – Example of an evaluation net

When the described system is in state E, the arrival of request R1 or R2 results in a transition which switches it state F.

On the other hand, crossing a transition takes place, by convention, in zero time.

3.6.3 Simple actions

Simple actions can be associated with each transition, corresponding either to the transmission of requests or to the execution of local actions.

They are materially represented by a triangle, labelled with the name of the request or of the description of the executed action.

3.6.4 Conditions

The crossing of a transition can concern a condition. In this case, the oriented graph associated with the representation, shows a condition associated with the arc which precedes the transition.

A corollary of this representation allows defining choices for crossing a transition from the same initial state.

In the same way as for a simple transition, the transmission of an action can be associated with the crossing of a conditional transition.

4 Abstract syntax of data type

4.1 Data abstract syntax specification

This present standard uses the following terms defined in the abstract syntax number 1 (ASN1). Different or added definitions are the following :

4.1.1 Boolean

A Boolean value is encoded on an octet all of whose bits are null for FALSE; any other combination means TRUE.

Example of notation of the type:

Response::= BOOLEAN

Example of value notation and encoding:

Response value::= TRUE

value is encoded, e.g., FFh or 6Dh.

4.1.2 Integer

The INTEGER value is encoded as a sequence of M octets describing a binary number:

- either as a complement of 2 if the values can be positive and negative: the most and least significant bits are respectively bit 8 of the first octet and bit 1 of the last octet. Bit 8 of the first octet represents the integer sign. If the size is the number of bits of the integer (i.e., $\text{size} = M \times 8$), the value of the integer is between $-2^{(\text{size}-1)}$ and $+2^{(\text{size}-1)}-1$;

Example of notation of the type:

Signed8::= INTEGER (-128..127)

Example of value notation and encoding:

integer Signed8::= -128

integer is encoded: FFh.

- or as an absolute value if the values are all positive or all negative: the most and least significant bits are respectively bit 8 of the first octet and bit 1 of the last octet. If the size is the number of bits of the integer (i.e., $\text{size} = M \times 8$), the value of the integer is between $-2^{(\text{size})}$ and $+2^{(\text{size})}-1$.

Example of type notation:

Unsigned8::= INTEGER (0..255)

Example of value notation and encoding:

integer Unsigned8::= 255

integer is encoded: FFh.

4.1.3 Bit string

The value of BIT STRING is encoded on a sequence of M octets; the first bit of the binary string is aligned with bit 8 of the least significant octet of the sequence of octets and its number is zero. The number of unused bits shall be between bit 1 and 7 of the most significant octet of the sequence of octets. The true value of a bit of the bit string takes the binary value 1; unused bits can take the value 0 or 1.

Example of notation of the type:

Bitstring::= BIT STRING SIZE(32)

Example of value notation and encoding:

binaryvalue Bitstring::= '5F291CD0'H

binary value is encoded: 5Fh 29h 1Ch D0h

4.1.4 Octet string

The value of OCTET STRING is encoded on a sequence of M octets; the first octet of the OCTET STRING is aligned with the least significant octet of the sequence of octets.

Example of type notation:

OctetString::= OCTET STRING SIZE(4)

Example value notation and encoding:

octetvalue OctetString ::= '5F291CD0'H

octetvalue is encoded: 5Fh 29h 1Ch D0h.

4.1.5 Sequence (sequence or sequence of)

SEQUENCE or SEQUENCE OF encoding is the juxtaposition of the element encoding which comprise it.

Example of type notation:

INFO1 ::= SEQUENCE {name String, ok BOOLEAN}

String ::= OCTET STRING SIZE(5) --string of 5 octets

Example of value notation and encoding:

value INFO1 ::= {name "SMITH", ok TRUE}

value is encoded as shown in Table 1.

Table 1 – Example of encoding of a SEQUENCE

0006h	53h	4Dh	49h	54h	48h	FFh
SEQUENCE length	"S"	"M"	"I"	"T"	"H"	TRUE

The top line represents the encoding proper and the bottom line describes what is encoded:

Example of type:

INFO2 ::= SEQUENCE OF date

date ::= OCTET STRING SIZE (8) --YYYYMMDD

Example of value notation and encoding:

value INFO2 ::= { { date "19571111"}, {date "19590717"} }

value is encoded as shown in Table 2.

Table 2 – Example of encoding of a SEQUENCE OF

00h	10h
number	of octets

31h	39h	35h	37h	31h	31h
"1"	"9"	"5"	"7"	"1"	"1"

31h	31h	31h	39h	35h	39h
"1"	"1"	"1"	"9"	"5"	"9"

30h	37h	31h	37h
"0"	"7"	"1"	"7"

In each row pair, the top line represents the encoding proper and the bottom line describes what is encoded. Each table is read from left to right.

4.1.6 Choice

The encoding of CHOICE is the encoding of the possibility selected in this CHOICE.

Example of type notation:

INFO::= CHOICE {name [0] String, age [1] Unsigned8}
 String::= OCTET STRING SIZE(5) --string of 5 octets
 Unsigned8::= INTEGER (0..127) -- unsigned 8 bit integer
 Example value notation and encoding:
 value INFO::= {name "SMITH"}
 value is encoded as shown in Table 3.

Table 3 – Example of encoding of a CHOICE

80h	53h	4Dh	49h	54h	48h
identification	"S"	"M"	"I"	"T"	"H"

The top line represents the encoding proper and the bottom line describes what is encoded:

4.1.7 Null

No content encoding is associated with the null element.

Identification encoding is possible if NULL is a possibility of a choice (Cf. section 10.1.2.5.2).

Example of notation of the type:

```
Type-Room::= SEQUENCE {
    number Unsigned8,
    person CHOICE {
        name [0] String,
        default [1] NULL --room not in use
    }
}
```

Unsigned8::= INTEGER (0..127)

String::= OCTET STRING SIZE(12)

Example of value notation and encoding:

value-Room Type-Room::= {number 30h, person { default NULL}}

value-Room is encoded 0002h 30h 81h

Length encoding is compulsory if NULL is optional (in this case, the length is necessarily equal to zero).

4.1.8 Object identifier

Encoding is comprised of an ordered list of encoding of concatenated sub-identifiers.

Every sub-identifier is represented by a sequence of one or more octets. Every octet is encoded as follows.

- Bit 8 of each octet indicates that it is the last in the sequence.
- Bit 8 of the last octet is on one.
- Bit 8 of each previous octet is on zero.
- Bits 7 to 1 of the octets in the sequence collectively encode the sub-identifier, knowing that:

Bits 5, 6 and 7 are zero.

- Bits 1 to 4 are used to encode each digit of each sub-identifier. Encoding occurs in the form of an unsigned binary number.

Example: {iso standard 9506 part(2) mms-general-module-version1(2)} is encoded as shown in Table 4.

Table 4 – Example of encoding of an object identifier

81h	80h	09h	05h	00h
"iso"	"standard"	9	5	0

86h	82h	82h
6	part(2)	mms...(2)

In each table, the top line represents the encoding proper and the bottom line describes what is encoded. Each table is read from left to right.

4.1.9 Default

The keyword DEFAULT is ignored.

4.2 FAL PDU abstract syntax

This section illustrates the FER encoding rules specified in the present standard, by showing the representation in octets of an element defined in FIELD BUS ASN.1.

```

PDU ::= CHOICE {
    req [0] IMPLICIT PDUreq,
    rep [1] IMPLICIT PDUrep, ....
}

PDUrep ::= SEQUENCE {
    invokeID Unsigned32,
    Response }

Response ::= CHOICE {
    status [0] IMPLICIT StatusResponse, ...
    getprog [45] IMPLICIT GetProgramInvocationAttributesResponse, ...
}

GetProgramInvocationAttributesResponse ::= SEQUENCE {
    pi_state [0] IMPLICIT PiState,
    listofDomainId [1] IMPLICIT ListOfObjectID,
    mmsdeletable [2] IMPLICIT BOOLEAN,
    reusable [3] IMPLICIT BOOLEAN,
    monitor [4] IMPLICIT BOOLEAN,
    executionargument [5] IMPLICIT OCTET STRING OPTIONAL
}

PiState ::= Unsigned8

Unsigned8 ::= INTEGER (0..127) --8 bit unsigned integer

ListOfObjectID ::= SEQUENCE OF Identifier

Identifier ::= Unsigned16

Unsigned16 ::= INTEGER (0..32767) --16 bit unsigned integer

Unsigned32 ::= INTEGER (0..2147483647) --32 bit unsigned integer

The registered value is:
{ rep { invokeID 1,
    getprog { pi-state 8,
    listofDomainId { {1024}, {1025}, {1026}, {1027} },
    mmsdeletable TRUE,
    reusable FALSE,
    monitor TRUE,
    executionargument "ARGUMENT" }
}

```

Its representation is as follows: octet encoding is described using the two-line tables below in which the top line represents the encoding proper and the bottom line describes what is encoded. The tables are read from left to right. See Table 5 for details.

Table 5 – Example of encoding of a PDU

81h	00h	00h	00h	01h	ADh		
rep	invokeID	(32 b ...		...)	getprog		

08h	00h	08h	04h	00h	04h	01h	
pi-state	number	of octets	1024	(16 b)	1025	(16b)	

04h	02h	04h	03h	FFh	00h	FFh	00h
1026	(16b)	1027	(16b)	TRUE	FALSE	TRUE	number

08h	41h	52h	47h	55h	4Dh	45h	4Eh
of octets	"A"	"R"	"G"	"U"	"M"	"E"	"N"

54h	
"T"	

5 Transfer syntaxes

5.1 Compact encoding

The encoding rules applied to the PDU type values represented by using the ASN1 (ISO 8824) abstract notation syntax, yield the transfer syntax.

In the MPS environment, the values associated with the PDU types, are globally encoded according to the encoding rules of ASN1 (ISO/IEC 8825), with some restrictions.

The encoding rules which yield the MPS transfer syntax associate a set of octets with each of the different PDU types.

This octet sequence comprises three components:

- identification
- size
- contents.

In the general case, the encoding rules permit contents which are recursively structured themselves in Identification, Size, Contents.

NOTE In the case of a CompactValue type PDU, the encoding has only one level.

The organization of an encoding level corresponds to that of the ISO/IEC 8825.

Encoding structure is shown in Figure 2.

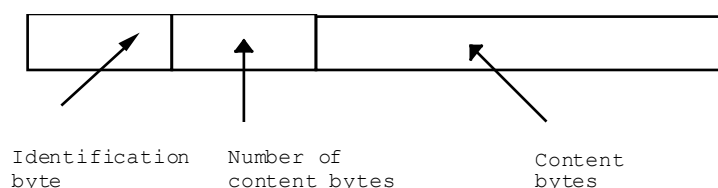


Figure 2 – Encoding of a CompactValue

The following restrictions on the encoding relative to the MPS standard apply:

The Identification is encoded on 1 octet.

The Contents is encoded on a maximum of 126 octets.

5.2 Data type encoding

5.2.1 MPS ASE FAL PDU data types

5.2.1.1 Summary

The specification of the protocol provides a set of communication rules expressed in terms of data interchanges and procedures that shall be supported by the application entities.

AL TYPE 7 conformance is based on the compliance to these communication rules which are guidelines for conformance testing in order to achieve interoperability.

This protocol specification consists of:

- the description of the various protocol data units (PDU) interchanged between the communicating entities. This description is affected by use of an abstract syntax.
- encoding rules which transform the abstract syntax into transfer syntax.
- the description of the protocol procedures in terms of interactions between the primitives of each of the application services and the associated data link service primitives.

NOTE In the MPS environment, there is no direct interaction between a producer application entity and a consumer entity during the interchanges associated with a service primitive.

The specification of A_PDUs in the MPS environment is performed with a notation defined independently from the encoding. This notation constitutes the abstract syntax.

The abstract syntax is subsequently transformed by a set of encoding rules which are applied to the various types of A_PDU. These encoding rules yield the transfer syntax.

The conjunction of the abstract syntax and the encoding rules specifies the content of the octets interchanged between application entities by using the data link services.

NOTE The actual representation at the interface between the user and an application entity is out of the scope of this standard and is implementation dependent.

5.2.1.2 Syntax notation

The specification of the different A_PDUs associated with the MPS services is performed with the abstract notation syntax ASN1 defined in the ISO/IEC 8824.

All the ASN1 definitions of this standard are part of the module: " AL TYPE 7 - -MPS-1 ". All the grammar elements which constitute the Part5-3-MPS-1 module correspond to the first version of MPS.

Declaration of the AL TYPE 7 - -MPS-1 module:
AL TYPE 7 - -MPS-1 DEFINITIONS:= BEGIN

NOTE In order to meet the requirements of the ASN1 notation, all the keywords of the language are written in capital letters and the final symbol names begin with a capital letter. Furthermore, all the grammar items existing in this standard which take part in the definition of the AL TYPE 7 - -MPP-1 module are outlined with a double line before and after them.

General quote: The present standard specifies the value of each octet of an encoding by the sentences 'the most significant bit' and 'the least significant bit'. The bits of an octet are numbered from 8 to 1: bit 8 is 'the most significant bit' and bit 1 'the least significant bit'. This encoding is summarized in Figure 3.

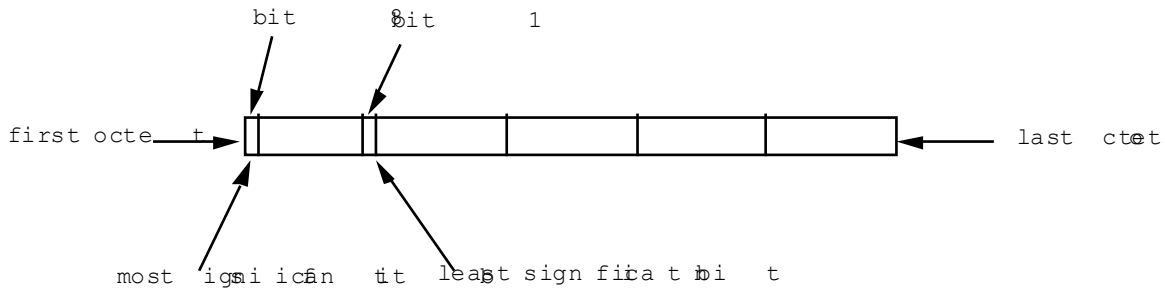


Figure 3 – Organisation of the bits and octets within a PDU

5.2.1.3 Simple predefined types

The PDU specification uses simple predefined types.

Some simple types are predefined by means of a statement that defines limits for these type values.

Unsigned4	::= INTEGER	-- values limited between 0 and 15
Unsigned8	::= INTEGER	-- values limited between 0 and 255
Unsigned16	::= INTEGER	-- values limited between 0 and 65535
Symbol	::= Visible String	-- number of characters limited between 0 and 16

In the MPS environment the variable names, the access names, the type names and the list names are described as a combination of 16 character maximum length.

The characters belong to the set defined for the 'visible string' type of the ASN1 (ISO 8824) syntax notation.

In this set, the following characters are permitted:

Capital letters	(A..Z)
Letters	(a..z)
Digits	(0..9)
Underscore	(_)
Currency symbol (\$)	

The 'space' character is not permitted.

Strings may not begin with a number.

The 'Unsigned16', 'Unsigned8' and 'Unsigned4' types are used in this standard to describe the limits of the maximum values of the MPS types.

5.2.1.4 PDU Types

5.2.1.4.1 Top Level Definition

The PDUs specified in the MPS environment convey the information associated with *Variable* objects. The PDU type used depends directly on the value of the *Construction* and *A_Name* attributes of the instance of the *Type constructor* associated with the variable and is summarized in Table 6.

Table 6 – MPS PDU types

Construction		PDU type
SIMPLE		CompactValuePDU
ARRAY		CompactValuePDU
STRUCTURE		CompactValuePDU
PREDEFINED	A_Name = K_DVAR	VariableDescriptionPDU
	A_Name = K_DACCESS	AccessDescriptionPDU
	A_Name = K_DTYPE	TypeDescriptionPDU
	A_Name = K_DLIST	ListDescriptionPDU
EXPLICIT		ExplicitPDU

The encoding of values of these different kinds of PDU uses the transfer syntax associated with ASN1 part 2 (ISO 8825). Nevertheless, array compacting and compressing rules can be implemented to optimize the flow on the bus.

NOTE These compacting rules allow the representation of structured type variables by using an octet string.

Specification of PDU type used in MPS:

MPS-pdu:: = CHOICE

```
{
  CompactedVariableValue      [APPLICATION 0]  IMPLICIT CompactValuePDU,
  ExplicitVariableValue       [APPLICATION 1]  IMPLICIT ExplicitPDU,
  VariableDescription         [APPLICATION 2]  IMPLICIT VariableDescriptionPDU,
  AccessDescription           [APPLICATION 3]  IMPLICIT AccessDescriptionPDU,
  TypeDescription             [APPLICATION 4]  IMPLICIT TypeDescriptionPDU,
  ListDescription             [APPLICATION 5]  IMPLICIT ListDescriptionPDU,
  extensions                  [APPLICATION 6]  ANY
}
```

NOTE The [APPLICATION 7] to [APPLICATION 15] type PDUs are reserved by this standard for a future definition of other PDU types. The [APPLICATION 16] to [APPLICATION 30] type PDUs are reserved for the use of the Network Management standard. Moreover, the ANY type PDU is used to allow the future definition of additional PDU types in companion standards.

5.2.1.4.2 VariableCompactValue PDU

5.2.1.4.2.1 General

This PDU represents the variable value of SIMPLE, STRUCTURE or ARRAY construction type, and optionally the associated production status. These types are used for variables of NORMAL or SYNCHRONIZATION class.

The PDU of CompactValuePDU type is represented by using the OCTET STRING type:

Compact-value-pdu:: = OCTET STRING

-- The total length of the PDU shall not exceed 128 octets.

NOTE The fields of this PDU permits encoding of the following attributes in the MPS application layer objects. See Table 7.

Table 7 – Fields of a CompactValuePDU

Field	Class	Attribute
Compact_value_PDU	Variable	Public value Transmitted status value

The ASN1 OCTET STRING type will be then encoded under the ASN1 (ISO 8825 standard) rules as specified in the chapter concerning the transfer syntax.

NOTE The representation of a MPS variable under an ASN1 OCTET STRING type is only possible considering the following restrictions.

- Producers and consumers of the variable have a prior knowledge of the MPS type of the variable, and do not need to transmit the type description with the value of the variable. The knowledge is available by access to the description variable and description type variables.
- The type of the variable has a fixed length and has no conditional or optional fields.

The representation of the public value associated with a variable by using compacting rules is performed from the first content octet of the CompactValuePDU.

The production status are represented on the last octet when this option is selected.

The PDU's length octet value represents the octet count used to encode the variable value and the production status octet when it exists.

The representation of the production status on this octet is performed as follows.

- The refreshment status is encoded on bit 1. This bit is set to '1' when the refreshment status is 'TRUE', and '0' when it is 'FALSE'.
- The punctual refreshment status is encoded on bit 2. This bit is set to '1' when the punctual refreshment status is 'TRUE', and '0' when it is 'FALSE'.
- The bits 3 to 8 are reserved.

The encoding of a given variable is represented with a constant number of octets derived from the type specification of this variable.

5.2.1.4.2.2 BOOLEAN value encoding rule

The boolean encoding is performed on a single octet.

If the value is

FALSE: the octet is 00₁₆

TRUE: the octet may take any value different from zero chosen by the producer entity.

5.2.1.4.2.3 INTEGER value encoding rule

The encoding of an integer value is performed on one or several octets. The maximum number of octets is fixed by the maximum size authorized for the INTEGER type, that is, 32 octets.

The number of octets used for encoding an INTEGER type corresponds to $E[(e-1)/8] + 1$ where E is the integer part, and e the value of the *size* attribute associated with this type.

The content octets shall be a two-complement binary number equal to the integer value and be composed of bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet, followed by bits 8 to 1 of each following octet, up to and including the last octet of the contents.

NOTE The value of a twos-complement binary number is derived by numbering the bits of the content octets, starting from bit 1 of the last octet and ending at bit 8 of the first octet. Each bit is assigned a numerical value of 2^{n-1} , where n is the bit position within the numbering sequence. The value of the twos-complement binary number is obtained by adding the numerical values assigned to each of the bits set to '1' (except for bit 8 of the first octet) from which the signed value from bit 8 of the first octet. is derived. The sign is negative when bit 8 of the first octet is set to '1'.

5.2.1.4.2.4 BITSTRING value encoding rule

The encoding of a bitstring value is performed on one or several octets as shown in Figure 4.

The maximum number of octets is fixed by the maximum size authorized for a BITSTRING type, that is, 32 octets.

The number of octets used for encoding a BITSTRING type corresponds to $E[(e-1)/8] + 1$ where E is the integer part and e the size of this type.

The bits of the bit string, starting from the first bit and ending at the last bit, shall be placed in sequence and in this order:

- in bits 8 to 1 of the first octet for the first 8 bits of the string,
- in bits 8 to 1 of the second octet for the 8 following bits,
- then in bits 8 to 1 of each following octet,
- followed by as many bits of the final octet as necessary always starting with bit 1.

The unused bits of the last octet have a value chosen by the producer entity.



Figure 4 – Encoding of a Bitstring

5.2.1.4.2.5 UNSIGNED value encoding rule

The encoding of an unsigned integer value is performed on one or several octets. The maximum number of octets is fixed by the maximum size permitted for an UNSIGNED type, that is, 32 octets. The number of octets used for encoding a UNSIGNED type corresponds to $E[(e-1)/8] + 1$ where E is the integer part and e the value of the *size* attribute associated with this type.

The content octets shall be a binary number equivalent to the integer value, composed of bits 8 to 1 of the first octet, followed by bits 8 to 1 of the second octet, followed by bits 8 to 1 of each following octet, up to and including the last octet of the contents.

NOTE The value of a binary number is obtained by numbering the bits of the content octets, starting from bit 1 of the last octet and ending at bit 8 of the first octet. Each bit is assigned a numerical value of 2^{n-1} , where n is the bit position within the numbering sequence.

5.2.1.4.2.6 OCTETSTRING value encoding rule

The encoding of an OCTETSTRING value contains zero, one or several content octets whose value is equal to that of the data value octets, in the same sequence as they are placed in the data value, the most significant bit of a data value being aligned with the most significant bit of an octet of the contents.

The number of octets used for encoding an OCTETSTRING type corresponds directly to the number of terms of this string within the limit of 126 octets.

5.2.1.4.2.7 VISIBLE STRING value encoding rule

The value of a visible string is composed of a string of characters from the set specified in ISO 8824. Each of the characters of this string is encoded in one octet.

The number of octets used for encoding a VISIBLE STRING type corresponds directly to the number of elements of the string within the limit of 126 octets.

5.2.1.4.2.8 GENERALIZED TIME value encoding rule

The encoding of the value is performed in the same manner as a 14 character VISIBLE STRING value.

The generalized time (specified in ISO 8824) is characterized by a specific number of characters encoded on octets.

The generalized time is a juxtaposition of characters in the following sequence:

- 4 digits for the year
- 2 digits for the month
- 2 digits for the day
- 2 digits for the hour
- 2 digits for the minutes
- 2 digits for the seconds.

The first octets are those including the encoding of the year.

The generalized time expresses directly a local time.

5.2.1.4.2.9 FLOATING POINT value encoding rule

The encoding of a floating point value is performed on 4 or 8 octets according to the size attribute corresponding to single or double precision.

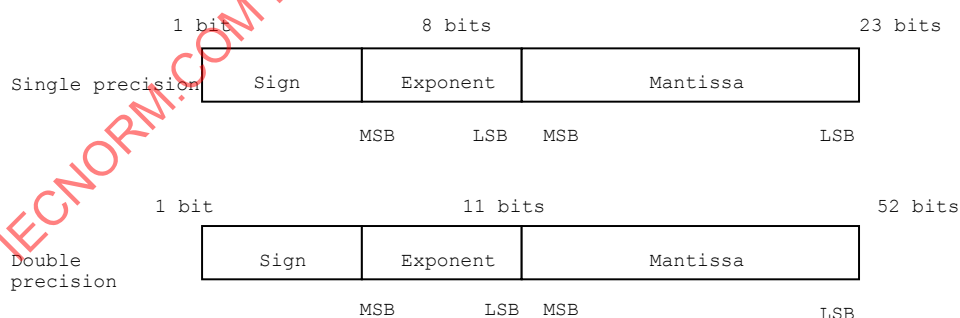


Figure 5 – Encoding of a Floating point value

The presentation formats defined in IEC 60559 are shown in Figure 5.

The content octets contain the sign, the exponent and the mantissa in the previous order starting from the first to the last octet.

In all cases the sign is encoded by using bit 8 of the first octet. It is followed by the the exponent starting from bit 7 of the first octet, and then the mantissa starts from bit 7 of the second octet for the simple precision and from bit 4 of the second octet for the double precision.

The encoding of the mantissa value and the exponent value is described in IEC 60559.

5.2.1.4.2.10 BINARY TIME value encoding rule

The encoding of a binary time value is performed on a number of octets defined by the size, and describes a number of elementary time units. The value of the elementary time unit depends on the selected size.

The content octets contain a binary value equivalent to the binary time value, and composed of bits 8 to 1 of the first octet, followed by bits 8 to 1 of each following octet, up to and including the last octets of the contents.

NOTE The binary value is obtained by numbering the bits of the content octets, starting from bit 1 of the last octet and ending at bit 8 of the first octet. Each bit set to '1' is assigned a numerical value of 2^{n-1} , where n is the bit position within the numbering.

5.2.1.4.2.11 BCD value encoding rule

The BCD encoding is performed on one octet. The binary value associated with the BCD value is encoded on bits 1 to 4 of the octet and obtained by starting from bit 1 up to bit 4. Each bit set to '1' is assigned a numerical value of 2^{n-1} , where n is the position of the bit within the octet.

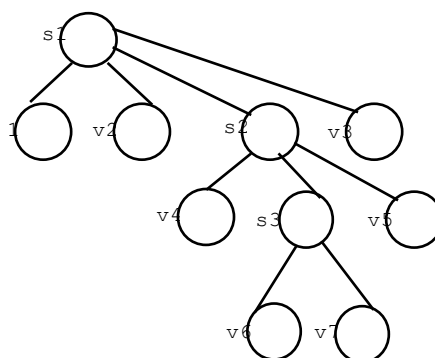
5.2.1.4.2.12 Structure encoding rule

The encoding of a structure type involves the encoding rules of each of the simple types and a concatenation rule of the types composing this structure.

The encoding of the various types composing a structure are juxtaposed in order of appearance within the structure.

The value associated with the first structure field is encoded in the first octets of the content. The value associated with the second field is encoded in the following octets, and so on, up to the last field of the structure.

Consider the following type tree:



Representation within the octets of the contents:

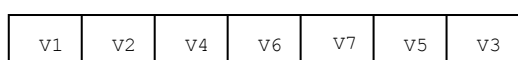


Figure 6 – Encoding of a structure

A structure can comprise several levels of abstraction; in this case, the encoding juxtaposition of the values associated with the various fields is performed by a first in depth traverse over the type tree associated with the structure as shown in Figure 6.

5.2.1.4.2.13 Array encoding

The array encoding involves the encoding rule of a simple type and a concatenation rule of the elements which compose the array.

The encoded values of the array elements are juxtaposed in the statement sequence of the array: these elements may be of simple, structure or array type.

The first array element is contained in the first octet of the content octets.

The array types may be optionally specified with optimisation rules of the associated encoding.

These optimisation rules do not preclude a partial access to an array element and preserve the integrity of the handled data.

5.2.1.4.2.14 Compacted encoding rule for a BOOLEAN array

The normal encoding of a BOOLEAN array type value uses one octet for each element of the array.

The compacted encoding of a BOOLEAN array type value is performed using one octet for a set of eight booleans.

Thus, the content octets used for encoding a compacted BOOLEAN array are organized as shown in Figure 7.

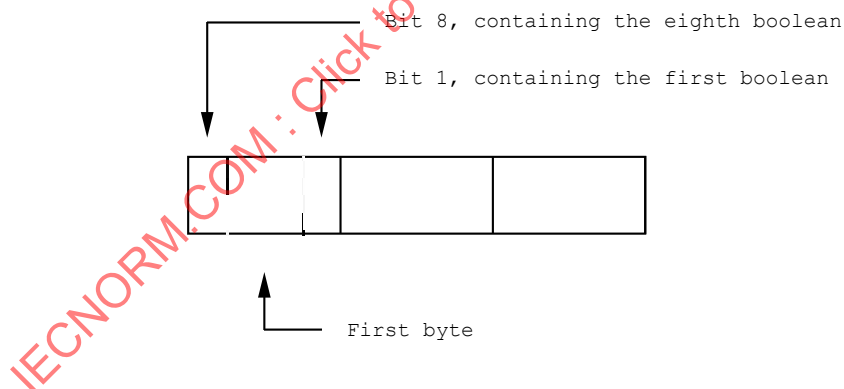


Figure 7 – Encoding of a Boolean array

In the compacted encoding of a boolean array, a boolean value is 'TRUE' when the corresponding bit is set to '1'.

Every compacted boolean array is encoded using a number of octets equal to $E[(e-1)/8] + 1$ where E is the integer part and e is the number of elements of this boolean array.

5.2.1.4.2.15 Compacted encoding rule for a BCD array

The normal encoding of a BCD array type value uses one octet for each element of the array.

The compact encoding of a BCD array type value is performed by using one octet for a pair of BCDs.

Thus, within the content octets used for encoding a compacted BCD array, bits 8 to 5 of the first octet contain the first BCD of the array, and bits 4 to 1 of the first octet contain the second BCD of the array, and so on up to the last BCD of the array.

The binary value associated with the value of the first BCD of a octet is obtained by starting from bit 5 up to bit 8: each bit is assigned a numerical value of 2^{n-5} , where n is the position of the bit within the octet.

The binary value associated with the value of the second BCD of a octet is obtained by starting from bit 1 up to bit 4: each bit is assigned a numerical value of 2^{n-1} , where n is the position of the bit within the octet.

5.2.1.4.3 ExplicitVariableValue PDU

The Explicit PDU type PDU allows the representation of the values of the variables for which the type construction is EXPLICIT.

This PDU supports variable values which are not of constant length.

ExplicitPDU ::= CHOICE

{	
array	[0] IMPLICIT SEQUENCE OF ExplicitPDU,
structure	[1] IMPLICIT SEQUENCE OF ExplicitPDU,
boolean	[2] IMPLICIT BOOLEAN,
bitString	[3] IMPLICIT BITSTRING,
integer	[4] IMPLICIT INTEGER,
unsigned	[5] IMPLICIT INTEGER,
floating	[6] IMPLICIT OCTET STRING,
octetString	[7] IMPLICIT OCTET STRING,
visibleString	[8] IMPLICIT OCTET STRING,
generalizedTime	[9] IMPLICIT OCTET STRING,
binaryTime	[10] IMPLICIT OCTET STRING,
bcd	[1] IMPLICIT INTEGER
}	

The total length of the PDU shall not exceed 128 octets.

5.2.1.4.4 VariableDescription PDU

This PDU represents the value of the variables whose type is K_DVAR. This type is used for the description variables.

VariableDescriptionPDU ::= SEQUENCE

{	
FixedDescription	[0] IMPLICIT Description, -- predefined type specified hereafter
variableName	[1] IMPLICIT Symbol,
resynchronisationReference	[2] IMPLICIT ObjectReference OPTIONAL,
productionPeriod	[3] IMPLICIT Unsigned16 OPTIONAL,
synchronisationReference	[4] IMPLICIT ObjectReference OPTIONAL,
ProductionSlot	[5] IMPLICIT Unsigned16 OPTIONAL,
punctualSynchroReference	[6] IMPLICIT ObjectReference OPTIONAL,
listReference	[7] IMPLICIT ObjectReference OPTIONAL,
otherDetails	[8] IMPLICIT ANY OPTIONAL,
vendorInformation	[9] IMPLICIT OCTET STRING OPTIONAL
}	

The total length of the PDU shall not exceed 128 octets. See Table 8.

NOTE The fields of this PDU permits encoding of the following attributes in the MPS application layer objects.

Table 8 – Fields of a VariableDescriptionPDU

Field	Class	Attribute
FixedDescription	Variable	Identifier Network Period Class Consistency Variable Significant Status Reference Type Constructor
Variable Name	Variable	A_Name
ResynchronizationReference	Resynchronization in Production	Reference (Synchronization) Variable
ProductionPeriod	Refreshment	Production Variable
SynchronizationReference	Refreshment	Reference (Synchronization) Variable
ProductionSlot	Punctual Refreshment	Production Time Slot
PunctualSynchroReference	Punctual Refreshment	Reference (Synchronization) Variable
ListReference	Variable	Reference Variable List

The type description of this PDU uses the Description type which contains the non optional part of constant size of this PDU.

Description:: = OCTET STRING -- Size fixed to 7 octets

Octet semantics:

octets 1 + 2: Described variable identifier

(Most significant bit of the identifier in bit 8 of the first octet and least significant bit in bit 1 of the second octet)

octets 3 + 4: Network period

(Most significant bit of the network period in bit 8 of the first octet, in multiples of 1 millisecond; a 0 value indicates an aperiodic variable)

octet 5: Bit 8, and 7: Variable class

00: NORMAL

01: SYNCHRONIZATION

10: DESCRIPTION

All the other combinations are reserved

Bit 6: Sub-class of a synchronization variable

1: Consistency variable

0: Synchronization variable but not consistency variable

Bits 5 and 4: Significant status

00: no significant status

01: significant refreshment

10: significant punctual refreshment

11: significant refreshment and punctual refreshment

Bits 3 to 1 reserved.

octets 6 + 7: Type reference

(encoded on 16 bits by using the description variable identifier of this type. The most significant bit is bit 8 of the octet 6 and the least significant one bit 1 of the octet 7)

The type description of this PDU uses the non terminal grammar element ObjectReference referring to variables, lists and types by means of the identifier of the associated description variable. These identifiers are encoded on two octets: the most significant bit of the identifier is encoded in bit 8 of the first octet and the least significant one in bit 1 of the second octet.

ObjectReference ::= OCTET STRING -- Size fixed to 2 octets

The ANY types included in the definition of the PDU, as well as the reserved bits within this PDU, used to support, the description complements specified elsewhere.

Moreover, optional octet strings permit, insertion of vendor specific information into this PDU.

5.2.1.4.5 AccessDescription PDU

This PDU supports the variable value whose type is K_DACCESS. This type is used for the access description variables. See Table 9.

AccessDescriptionPDU ::= SEQUENCE

```
{
  accessName          [0] IMPLICIT Symbol,
  rootReference       [1] IMPLICIT ObjectReference,
  accessPath          [2] IMPLICIT SEQUENCE OF CHOICE
  {
    structureFieldName [0] IMPLICIT Symbol,
    arrayElementIndex  [1] IMPLICIT Unsigned8
  } OPTIONAL
  otherDetails        [3] IMPLICIT ANY OPTIONAL,
  vendorInformation    [4] IMPLICIT OCTET STRING OPTIONAL
}
```

NOTE The fields of this PDU permits encoding of the following attributes in the MPS application layer objects.

Table 9 – Fields of an AccessDescriptionPDU

Field	Class	Attribute
Access Name	Access	A_Name
Root Reference	Access	Reference Variable
Access Path	Type Constructor	Field Name

The ANY types include description complements specified elsewhere in the definition of the PDU.

Moreover, optional octet strings permit insertion of vendor specific information into this PDU.

5.2.1.4.6 TypeDescription PDU

This PDU supports the variable value whose type is K_DTYPE. This type is used for the type description variables.

TypeDescriptionPDU ::= SEQUENCE

```

{
  typeName                               [0] IMPLICIT Symbol,
  typeDescription                         [1] CHOICE
  {
    array                               [0] IMPLICIT SEQUENCE
    {
      compressed                        [0] IMPLICIT BOOLEAN DEFAULT FALSE,
      dimension                         [1] IMPLICIT Unsigned8,
      elementReference                  [2] IMPLICIT ObjectReference
    },
    structure                            [1] IMPLICIT SEQUENCE OF SEQUENCE
    {
      fieldName                         [0] IMPLICIT Symbol OPTIONAL,
      fieldTypeReference                [1] IMPLICIT ObjectReference
    },
    -- simple                            Size                            Type Class
    boolean                            [2] IMPLICIT NULL,                    -- BOOLEAN
    binaryString                       [3] IMPLICIT Unsigned8, -- BITSTRING
    integer                            [4] IMPLICIT Unsigned8, -- INTEGER
    unsigned                           [5] IMPLICIT Unsigned8, -- UNSIGNED
    floatingPoint                      [6] IMPLICIT BOOLEAN, -- FLOATING POINT
    octetString                        [7] IMPLICIT Unsigned8, -- OCTET STRING
    visibleString                      [8] IMPLICIT Unsigned8, -- VISIBLE STRING
    universalTime                      [9] IMPLICIT NULL, -- UNIVERSAL TIME
    binaryTime                         [10] IMPLICIT Unsigned4, -- BINARY TIME
    bcd                               [11] IMPLICIT NULL, -- BCD
    predefined                         [12] IMPLICIT NULL,
    explicit                          [13] IMPLICIT NULL,
    },
    otherDetails                       [2] IMPLICIT ANY OPTIONAL,
    vendorInformation                  [3] IMPLICIT OCTET STRING OPTIONAL
  }
}

```

-- The total length of the PDU shall not exceed 128 octets. See Table 10.

NOTE The fields of this PDU permits encoding of the following attributes in the MPS application layer objects.

Table 10 – Fields of a TypeDescriptionPDU

Field	Class	Attribute
TypeName	TypeConstructor	A_Name
Compressed	TypeConstructor	Compressed
Dimension	TypeConstructor	Dimension
ElementReference	TypeConstructor	TypeConstructor
FieldName	TypeConstructor	FieldName
FieldTypeReference	TypeConstructor	TypeConstructor
BinaryString	TypeConstructor	Size
Integer	TypeConstructor	Size
Unsigned	TypeConstructor	Size
FloatingPoint	TypeConstructor	Size
OctetString	TypeConstructor	Size
VisibleString	TypeConstructor	Size
BinaryTime	TypeConstructor	Size

The ANY types include description complements specified elsewhere in the definition of the PDU.

Moreover, optional octet strings permit insertion of information specific to the vendors into this PDU.

ListDescription PDU

This PDU supports the variable value whose type is K_DLIST. This type is used for the list description variables.

```

ListDescriptionPDU ::= SEQUENCE
{
    listName          [0] IMPLICIT Symbol,
    variableReferenceList [1] IMPLICIT ListComponents,
    inconsistencyDetection [2] IMPLICIT Unsigned8 OPTIONAL,
    -- 0 UNDEFINED
    -- 1 PERMANENT
    -- 2 PERMANENT
    -- The other values are reserved
    synchronizationReference [3] IMPLICIT ObjectReference OPTIONAL,
    consistencyVariableList [4] IMPLICIT ListConsistencyVariables OPTIONAL,
    recoveryPeriod [5] IMPLICIT Unsigned16 OPTIONAL,
    otherDetails [6] IMPLICIT ANY OPTIONAL,
    vendorInformation [7] IMPLICIT OCTET STRING OPTIONAL
}

```

-- The total length of the PDU shall not exceed 128 octets. See Table 11.

NOTE The fields of this PDU permits encoding of the following attributes in the MPS application layer objects.

Table 11 – Fields of a ListDescriptionPDU

Field	Class	Attribute
ListName	VariableList	A Name
VariableReferenceList	VariableList	List of Reference ConsumedVariable
InconsistencyDetection	VariableList	InconsistencyDetection
SynchronizationReference	VariableList	Reference (Synchronization)Variable
ConsistencyVariableList	VariableList	List of Reference (Consistency) Variable
RecoveryPeriod	VariableList	RecoveryPeriod

The type description of this PDU uses the non terminal grammar element 'listComponents' which defines the representation format of the references to the variables which compose the described list.

ListComponents ::= OCTET STRING -- The size of this string depends on that of the list.

Each variable name which comprise this list is encoded on two octets of this octet string, starting from the first two octets up to the last two octets within the size limits provided by this PDU. The list variable names are represented on two octets in a unique way by means of the identifier of the description variable associated with each variable of this list. The most significant bit of the identifier is encoded in bit 8 of the first octet, and the least significant one in bit 1 of the second octet.

The type description of this PDU uses the non terminal grammar element 'listConsistencyVariables' which defines the representation format of the references to the consistency variables associated with the variable list in the case of the operation of the spatial consistency mechanism.

ListConsistencyVariables ::= OCTET STRING -- size relative to the number of application entities involved in the spatial consistency.

Each consistency variable name, which takes part in the elaboration of the spatial consistency of this list, is encoded on two octets of this octet string, starting from the first two octets up to the last two octets within the size limits provided by this PDU. The consistency variable names are represented on two octets in an unique way by means of the identifier of the description variable associated with each consistency variable.

The most significant bit of the identifier is encoded in bit 8 of the first octet, and the least significant one in bit 1 of the second octet.

5.2.1.4.7 PDU Extensions

The ANY types include description complements specified elsewhere in the PDU definition.

For, all the grammar elements participating in the definition of the module being defined, it is necessary to declare the end of this module.

END -- End of the module AL TYPE 7 - -MPS-1

5.2.2 MCS AR ASE AL PDU types

5.2.2.1 Coding rules

All MCS PDUs contain a whole number of octets. The octets of a PDU are numbered from 1 to a maximum of 256 for the longest PDUs. These octets are transmitted via the bus in order from the first octet to the last. Each of the octets of a PDU has bits numbered 1 to 8, where bit 1 is the least significant and bit 8 the first transmitted on the line. Finally, when a number of consecutive octets represents a binary number, the least significant octet contains the most significant value.

In the following figures, the least significant number octets are always on the left when represented horizontally, and at the top when represented vertically. Also, the bits in a octet are always represented with bit 8 on the left and bit 1 on the right.

5.2.2.2 Structure of the PDUs

5.2.2.2.1 Top Level Definition

As a general rule, all MCS PDUs consist, in order, of:

- the header containing:
- the header size,
- the fixed part containing what is mandatory,
- the variable part containing what is optional,
- the data, which are optionally present.

Representation is shown in Figure 8

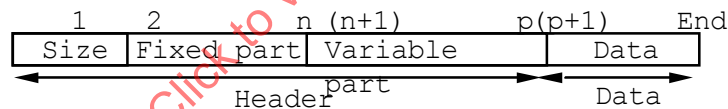


Figure 8 – Representation of a MCS PDU

a) Size:

The size is equal to sum of the numbers of octets of the fixed part and the variable part of the PDU header.

The value of the size is coded in the first octet of the header as a binary number.

The maximum size value is 255, which is coded as 1111 1111.

b) Fixed part

Whatever the PDU type, the fixed part contains its code followed by coding of a list of parameters which are mandatory for the code value.

The values of the PDU codes associated with the different types of PDUs are as shown in Table 12.

Table 12 – Coding of the different MCS PDU types

PDU values	Meaning
1000 0000	association establishment request
1100 0000	association establishment response
1001 0000	association termination request
1101 0000	association termination response
1011 0000	association revocation request
00X0 0XXX	associated transfer
0110 0000	associated transfer acknowledgement
00X1 0XXX	non-associated transfer
0111 0XXX	non-associated transfer acknowledgement

The bits of certain codes marked X have the values indicated in the descriptions of the associated PDU types.

NOTE All other codes are pending in this standard.

c) Variable part

This field is intended to receive optional parameters, which may be present or absent depending on the classes of conformity of the devices and the requirements of the user.

Each service parameter encoded in the variable part is structured as shown in Table 13.

Table 13 – Coding of the variable part of the PDU

Mapping	Meaning
octet 1	Parameter code
octet 2	Parameter length (length = m)
octet 2 + 1 (to)	Parameter value start (continued)
octet 2 + m	Parameter value end

Each of the parameters has a parameter code which is universal in the MCS environment, and which is therefore covered by this standard (these are defined for each PDU type implementing them).

The length of the parameter is coded in one octet. The value of the parameter is coded in as many octets as is specified in the length and the semantics of the coding is specific to each parameter (the semantics are defined at the level of each type of PDU implementing them).

d) Data

This field is transparent for MCS as it contains data specific to one of the specific application service elements projected onto MCS.

5.2.2.2.2 Association establishment request (AARQ)

AARQ structure is as shown in Table 14.

Table 14 – Structure of association establishment request

Mapping	Meaning
octet 1	Size
octet 2	PDU code
octet 3	Calling AEI access point
octet 4	ditto
octet 5	ditto
octet 6	Called AEI access point
octet 7	ditto
octet 8	ditto
octet 9	Class of conformity
octet 10	Variable part
to	ditto
octet p	ditto
octet p+1	User data
to	ditto
end octet	ditto

a) Size

The minimum size value is 8 (coded: 0000 1000) when the parameters relating to this PDU are not present, and the maximum size value is reached when all the parameters are present.

(For instance, the maximum size value is 51 octets.)

b) Fixed part

The following fields are present in the fixed part.

- The PDU code takes the value “1000 0000”.
- The calling AEI access point, which corresponds to a link address (of an ADAEI type) is represented by bit I/G as the high order bit in the lowest number octet.

Representation:

```

octet_3:  I / G x x x      x x x x
octet_4:  x x x x x x x x
octet_5:  S / R x x x      x x x x

```

- The called AEI access point which corresponds to a liaison address (of ADAEI type) is represented with the high order bit in the lowest number octet.

Representation:

```

octet_6 :  I / G x x x      x x x x
octet_7 :  x x x x x x x x
octet_8 :  S / R x x x      x x x x

```

NOTE The default value used for this access point when the association establishment initiator does not wish to fill it in is the following value:

```

octet_6 :      1111      1111
octet_7 :      1111      1111
octet_8 :      1111      1110

```

It is a link address value which is found in the generalised group address space reserved under the standard.

- The class of conformity

Representation:

octet_9: b8 b7 b6 b5 b4 b3 b2 b1

Where:

b8 = 1, if PV_RE = 2
 b7 = 1, if PH_AK = 2
 b6 = 1, if PH_RA = 2
 b5 = 1, if PH_SR = 2
 b4 = 1, if PH_CF = 2

The possible combinations of these bits shall allow for the constraints placed on the classes of conformity

c) Variable part

The following parameters shall be present in the variable part.

- Transfer rate:

Parameter code: 0000 0000

Parameter length: 4 octets (0000 0100)

Parameter value:

xxxx xxxx	:	Calling to Called
xxxx xxxx	:	Called to Calling
yyyy yyyy	:	
yyyy yyyy	:	

The value of these two rates is represented by a binary number equal to the time interval expressed in ms between successive messages.

The value 0000 0000 0000 0001 corresponds to 1msg/ms and the value 1110 1010 0110 0000 corresponds to 1msg/min.

The value 0000 0000 0000 0000 conventionally represents a null guaranteed minimum flow in one direction of transfer.

This parameter can only be present if b4 (PH_CF) of the Class of conformity parameter is set to 1.

- Anticipation factor:

Parameter code: 0000 0001

Parameter length: 2 octets (0000 0010)

Parameter value:

xxxx xxxx	:	Calling to Called
yyyy yyyy	:	Called to Calling

The anticipation factor is represented by a binary number making it possible to discriminate between 255 levels. The anticipation level of "i" means that it is possible to have a maximum of "i" ATAK_SM_RQ current in the association.

The use of value 0 is illegal.

This parameter can only be present if the b7(PH_AK) of the Class of conformity parameter is 1 .

- Application context:

Parameter code: 0000 0010

Parameter length: 3 octets (0000 0011)

Parameter value:

xxxx xxxx	:	Specific ASE type
yyyy yyyy	:	Abstract syntax type
zzzz zzzz	:	Transfer syntax type

The value of these different types is specific to each of the application ASEs. The high order bit of each of these octets indicates whether it is a type defined in a standard (b8=1) or specific to an application (b8=0).

- Calling entity identification:

Parameter code: 0000 0011

Parameter length: 6 octets (0000 0110)

Parameter value:

xxxx xxxx	:	AEI identifier
yyyy yyyy	:	API identifier
zzzz zzzz	:	AE qualifier
uuuu uuuu	:	AP title
uuuu uuuu		
uuuu uuuu		

The first octet of the value designates the AEI identifier by means of a binary number making it possible to discriminate between 255 AEIs for an API in the context of a given AE. Value 0000 0000 is the default value used to not fill in this parameter.

The second octet designates the API identifier by means of a binary number making it possible to discriminate between 255 APIs for a given AP. The value 0000 0000 is the default value used to not fill in this parameter.

The third octet designates the AE qualifier by means of a binary number making it possible to discriminate between 255 AEs for a given AP. The value 0000 0000 is the default value used to not fill in this parameter.

The last free octets designate the AP title by means of a binary number. These three octets are set to 0000 0000 if this parameter is not adopted.

- Called entity identification:

Parameter code: 0000 0100

Parameter length: 6 octets (0000 0110)

Parameter value:

xxxx xxxx	:	AEI identifier
yyyy yyyy	:	API identifier
zzzz zzzz	:	AE qualifier
uuuu uuuu	:	AP title
uuuu uuuu		
uuuu uuuu		

The first octet of the value designates the AEI identifier by means of a binary number making it possible to discriminate between 255 AEIs for one API in the context of a given AE. The value 0000 0000 is the default value used to not fill in this parameter.

The second octet designates the API identifier by means of a binary number making it possible to discriminate between 255 APIs for a given AP. The value 0000 0000 is the default value used to not fill in this parameter.

The third octet designates the AE qualifier by means of a binary number making it possible to discriminate between 255 AEs for a given AP. The value 0000 0000 is the default value used to not fill in this parameter.

The last three octets designate the AP title by means of a binary number. These three octets are set to 0000 0000 if this parameter is not adopted.

- SDU size:

Parameter code: 0000 0101

Parameter length: 2 octets (0000 0010)

Parameter value:

xxxx xxxx : Calling to Called

yyyy yyyy : Called to Calling

SDU size is represented by a binary number providing for a maximum SDU size of 256 DLPDUs, i.e. 64 octets for a maximum DLPDU size of 256 octets.

Value 0 of this parameter is illegal.

This parameter may only be present if b5(PH_SR) of the Class of conformity parameter is set to 1.

- PDU size:

Parameter code: 0000 0110

Length: 1 octet (0000 0001)

Parameter value:

xxxx xxxx : SDU size

PDU size is represented by a binary number allowing a maximum PDU size of 256 octets.

This parameter can be present whatever the value of the class of conformity.

- Number of retries:

Parameter code: 0000 0111

Length: 1 octet (0000 0001)

Parameter value: 0000 xxxx

The number of retries is represented by a binary number allowing a maximum of 16 retries coded in bits xxxx.

Value 0 of this parameter is illegal.

This parameter can only be present if b6(PH_RA) of the Class of conformity parameter is set to 1.

- “Cyclic flow” option:

Parameter code: 0000 1000

Length: 0 octet

d) Data

The data carried in this PDU is coded in a manner interpretable in the context of the application.

5.2.2.2.3 Association establishment response (AARP)

AARP structure is shown in Table 15.

Table 15 – Structure of an associated establishment response

Mapping	Meaning
octet 1	Size
octet 2	PDU code
octet 3	Calling AEI access point
octet 4	ditto
octet 5	ditto
octet 6	Called AEI access point
octet 7	ditto
octet 8	ditto
octet 9	Class of conformity
octet 10	Result
octet 11	Variable part
to	ditto
octet p	ditto
octet p+1	User data
to	ditto
end octet	ditto

a) Size

The minimum size value is 9 octets (coded 0000 1001), when the parameters relating to this PDU are not present and the maximum size value is reached when all the parameters are present.

b) Fixed part

The following fields are present in this part:

- The PDU code takes the value “1100 0000”.
- The calling AEI access point, which corresponds to a link address (of ADAEI type) is represented with bit I/G as the high order bit in the lowest number octet.

Representation:

```

octet_3:  I / G x x x   x x x x
octet_4:  x   x x x   x x x x
octet_5:  S / R x x x   x x x x

```

- The called AEI access point which corresponds to a link address (of ADAEI type), is represented with the high order bit in the lowest number octet.

Representation:

```

octet_6:  I / G x x x   x x x x
octet_7:  x   x x x   x x x x
octet_8:  S / R x x x   x x x x

```

NOTE The default value used for this access point when the association establishment initiator does not wish to fill it in is the following value:

```

octet_6: 1111   1111
octet_7: 1111   1111
octet_8: 1111   1110

```

It is a link address value which is found in the generalised group address space reserved under the standard.

- The class of conformity

Representation:

octet_9: b8 b7 b6 b5 b4 b3 b2 b1

Where:

b8 = 1, if PV_RE = 2
 b7 = 1, if PH_AK = 1
 b6 = 1, if PH_RA = 2
 b5 = 1, if PH_SR = 2
 b4 = 1, if PH_CF = 2

The possible combinations of these bits shall allow for the constraints on the classes of conformity.

- The result subsequent to an association establishment request is coded in one octet and indicated if the response is positive or negative, in the latter case providing information on the nature and location of the fault.

The values of the level 10 octet contain the following values in the following cases:

0000 0000: (+) Positive
 0000 0001: (-) MCS provider (general)
 0000 0010: (-) Responder user
 0000 0011: (-) linked to "Cyclic rate"
 0000 0100: (-) linked to "Transfer rate"
 0000 0101: (-) linked to "Context name"
 0000 0110: (-) linked to "Called entity identification"
 0000 0111: (-) linked to "Calling entity identification"
 0000 1000: (-) linked to "Service size"
 xxxx xxxx: Spare

c) Variable part

The following parameters can be present in the variable part.

- Transfer rate:

The coding and semantics of this parameter are equivalent to those of the parameter of the same name in the association establishment request.

- Anticipation factor:

The coding and semantics of this parameter are equivalent to those of the parameter of the same name in the association establishment request PDU

- Application context:

The coding and semantics of this parameter are equivalent to those of the parameter of the same name in the association establishment request PDU (see 10.2.3.2.2.c).

- Called entity identification:

The coding and semantics of this parameter are equivalent to those of the parameter of the same name in the association establishment request PDU

- SDU size:

The coding and semantics of this parameter are equivalent to those of the parameter of the same name in the association establishment request PDU (see 10.2.3.2.2.c).

- PDU size:

The coding and semantics of this parameter are equivalent to those of the parameter of the same name in the association establishment request PDU (see 10.2.3.2.2.c).

- Number of retries:

The coding and semantics of this parameter are equivalent to those of the parameter of the same name in the association establishment request PDU (see 10.2.3.2.2.c).

d) Data

The data carried in this PDU are coded in a manner interpretable in the context of the application.

5.2.2.2.4 Association termination request (RERQ)

The RERQ structure is shown as in Table 16.

Table 16 – Structure of an association termination request

Mapping	Meaning
octet 1	Size
octet 2	PDU code
octet 3	User data
to	ditto
end octet	ditto

a) Length

The size value is 1 octet (coded 0000 0001).

b) Fixed part

The single field containing the fixed part is the PDU code which takes the value “1001 0000”.

c) Variable part

There is no variable part in this PDU.

d) Data

Data carried in this PDU has coding which is interpretable in the context of the application.

5.2.2.2.5 Association termination response (RERP)

The RERP structure is as in Table 17.

Table 17 – Structure of an association termination response

Mapping	Meaning
octet 1	Size
octet 2	PDU code
octet 3	Result
octet 4	User data
to	ditto
end octet	ditto

a) Size

The size value is 2 octets (coded: 0000 0010).

b) Fixed part

The fixed part of the header contains the following two fields:

- PDU code: “1101 0000”
- The result which is coded in one octet, indicating whether the result is positive or negative, and in the second phase providing information on the fault, to be defined later.

0000 0000: (+) Positive
 0000 0001: (-) Negative
 xxxx xxxx: Spare (Defaults)

c) Variable part

This PDU has no variable part.

d) Data

The data carried in this PDU has coding which is interpretable in the context of the application.

5.2.2.2.6 Association revocation request (ABRQ)

The ABRQ structure is as in Table 18.

Table 18 – Structure of an association revocation request

Mapping	Meaning
octet 1	Size
octet 2	PDU code
octet 3	Invocation origin
octet 4	User data
to	ditto
end octet	ditto

a) Size

The size value is 2 octets (coded: 0000 0010).

b) Fixed part

The following two fields make up the fixed part:

- PDU code taking value “1011 0000”.
- Invocation origin, coded in one octet and taking two values:

0000 0000: MCS user
 1111 1111: MCS provider
 xxxx xxxx: Other codes reserved

c) Variable part

This PDU has no variable part.

d) Data

The data carried in this PDU has coding which is interpretable in the context of the application.

5.2.2.2.7 Associated transfer request (ATRQ)

The ATRQ structure is as in Table 19.

Table 19 – Structure of an associated transfer request

Mapping	Meaning
octet 1	Size
octet 2	PDU code
octet 3	Sequence number
octet 4	ditto
octet 5	User data
to	ditto
end octet	ditto

a) Size

The size value is 3 octets (coded: 0000 0011).

b) Fixed part

The following two fields constitute the fixed part of the header:

- PDU code, taking values:

0000 0000: for unacknowledged transfers

0010 0XXX: for acknowledged transfers

where XXX is used to code the packet nature in the case of segmentation.

- The packet nature is coded as follows:

000: NIL (no segmentation)

100: STT (first packet)

101: MID (middle packet)

110: END (last packet)

111: ALL (first and last packet).

- The sequence number is represented in two octets by means of a binary number. This makes it possible to discriminate between 65536 data transfers. This number is not significant in the case of unacknowledged transfer.

c) Variable part

This PDU has no variable part.

d) Data

The data carried in this PDU have coding which is interpretable in the context of the application specific to the association.

5.2.2.2.8 Associated transfer acknowledgement (AKAT)

The AKAT structure is as in Table 20.

Table 20 – Structure of an associated transfer acknowledgement

Mapping	Meaning
octet 1	Size
octet 2	PDU code
octet 3	Sequence number
octet 4	ditto

a) Size

The size value is 3 octets (coded: 0000 0011).

b) Fixed part

The following field is contained in the fixed part:

- PDU number, taking the value “0110 0000”.

- Sequence number, attributed by the numbering entity.

c) Variable part

This PDU has no variable part.

d) Data

This PDU has no user data.

5.2.2.2.9 Non-associated transfer request (NTRQ)

The NTRQ structure is as in Table 21.

Table 21 – Structure of a non-associated transfer request

Mapping	Meaning
octet 1	Size
octet 2	PDU code
octet 3	Invocation identification
octet 4	ditto
octet 5	Variable part
to	ditto
octet p	ditto
octet p+1	User data
to	ditto
end octet	ditto

a) Size

The minimum size value is 3 octets (coded: 0000 0011) when the parameters relating to the PDU parameter are not present, and the maximum value is reached when all the parameters are present.

b) Fixed part

The following fields are present in the fixed part:

- PDU code, taking values:

0001 0xxx: for unacknowledged transfers

0011 0xxx: for acknowledged transfers

where the value of xxx is used to code service priority.

- Service priority, coded in 3 bits making it possible to establish 8 priority levels.
- Invocation identification, represented in 2 octets by a binary number. This makes it possible to discriminate 65536 current simultaneous data transfers.

c) Variable part

The following parameters may be present in the variable part:

- Application context

The coding and semantics of this parameter are equivalent to those of the parameter with the same name in the association establishment request PDU (see 5.2.2.2.2).

- Calling entity identification

The coding and semantics of this parameter are equivalent to those of the parameter with the same name in the association establishment request PDU (see 5.2.2.2.2).

- Called entity identification

The coding and semantics of this parameter are equivalent to those of the parameter with the same name in the association establishment request PDU (see 5.2.2.2.2).

d) Data

The data carried in this PDU have coding which is interpretable in the context of the application.

5.2.2.2.10 Non-associated transfer acknowledgement (AKNT)

The AKNT structure is as in Table 22.

Table 22 – Structure of a non-associated transfer acknowledgement

Mapping	Meaning
octet 1	Size
octet 2	PDU code
octet 3	Invocation identification
octet 4	ditto

a) Size

The size value is 3 octets (coded: 0000 0011).

b) Fixed part

The following field is contained in the fixed part:

- PDU number, taking the value “0111 0XXX”.

The 3 bits XXX are reserved for coding a fault which can arise at the acceptor.

Fault coding is as follows:

000: AKNT+, no fault

≠000: AKNT-, fault

where:

001: non-specialised fault line

010 to 111: faults, the specialisation of which is reserved under this standard.

NOTE The acknowledgement does not carry the priority of the transfer request which gave rise to the acknowledgement as it is implicit that any requester will reserve the resources to receive the acknowledgement it awaits.

- Invocation identification is represented by 2 octets with a binary number. This makes it possible to discriminate 65536 current simultaneous data transfers.

c) Variable part

This PDU has no variable part.

d) Data

This PDU has no user data.

5.2.3 SUB MMS ASE FAL PDU type

5.2.3.1 SUB-MMS-PDU

5.2.3.1.1 Top Level Definition

This section describes the PDUs used in the SUB-MMS protocol.

SUB-MMS-General-module-1 {iso standard 9605
 part-protocol (2)
 sub-mms-general-module-version1 (5) }

```

DEFINITION:: = BEGIN
EXPORT SUB-MMSpdu;
-- By this affirmation we mean that any production subjected to SUB-MMSpdu is exportable.--
SUB-MMSpdu :: =
  confirmed-reqPDU          [0] IMPLICIT ConfirmedReqPDU,
  confirmed-respPDU         [1] IMPLICIT ConfirmedRespPDU,
  confirmed-errorPDU        [2] IMPLICIT ConfirmedErrorPDU,
  unconfirmedPDU            [3] IMPLICIT UnconfirmedPDU,
  rejectPDU                 [4] IMPLICIT RejectPDU,
  initiate-reqPDU           [5] IMPLICIT InitiateReqPDU,
  initiate-respPDU          [6] IMPLICIT InitiateRespPDU,
  initiate-errorPDU         [7] IMPLICIT InitiateErrorPDU,
  conclude-reqPDU           [8] IMPLICIT ConcludeReqPDU,
  conclude-respPDU          [9] IMPLICIT ConcludeRespPDU,
  conclude-errorPDU         [10] IMPLICIT ConcludeErrorPDU
}

```

The SUB-MMSpdu protocol defines 11 types of PDU

The companion Standards can increase this list.

5.2.3.1.2 Definition of confirmed-req-PDU

```

ConfirmedReqPDU:: =
  invoke-ID                UNSIGNED-32,
  confirmed-service-req     ConfirmedServiceReq}.

```

The confirmed-req-pdu includes two parameters:

- a) invoke-ID has the same significance as that given in the ISO/IEC 9506-2.
- b) ConfirmedServiceReq is defined below.

5.2.3.1.3 Definition of Unconfirmed-PDU

```

UnconfirmedPDU:: =
  invoke-ID                UNSIGNED-32,
  unconfirmed-service      UnconfirmedService}.

```

The unconfirmed-pdu includes two parameters:

- a) invoke-ID has the same significance as that given by the ISO/IEC 9506-2.

UnconfirmedService is defined below.

5.2.3.1.4 Definition of confirmed-resp-pdu

```

ConfirmedRespPDU:: =
  invoke-ID                UNSIGNED-32,
  confirmed-service-resp    ConfirmedServiceResp}.

```

The confirmed RespPDU includes two parameters:

The invoke-ID has the same significance as that given by the ISO/IEC 9506-2.

The ConfirmedServiceResp is defined below.

5.2.3.1.5 Definition of confirmed-error-pdu

```

ConfirmedErrorPDU:: =
  invoke-ID                UNSIGNED-32,
  service-error            ErrorType}.

```

The ConfirmedErrorPDU includes two parameters:

- a) invoke-ID has the same significance as that given by the ISO/IEC 9506-2.

Error is defined below.

5.2.3.1.6 Definition of "object" and "service"

5.2.3.1.6.1 Definition of a class of objects

A class of objects is a qualifying expression attributed to a type of set of objects to express the fact that the same action applied to these objects always gives the same result. We attribute a name to each class of objects. Consequently a class of objects shall be identified by its name. The CSs can increase this class of objects. See Table 23.

Table 23 – Definitions of object classes

Name of class	Type of object	Creation
vmd	visible part of the server	I/T
dom	domain	I/T/E
pi	executable program	I/T/E
var	variable	I/T
var-list	variables-list	I/T/E
event	event	I/T
association	connection	I/T/E
OD-header	OD-header	I/T
data type	type descriptor	I/T
LEGEND: I = Implicit creation. T = Creation following a remote loading. E = Explicit Creation by Service.		

5.2.3.1.6.2 Definition of a service

A type of action applied to an object belonging to a given class is called service. We attach a name to each of the services. Consequently, a service will be identified by its name. The CSs can increase this list of services. See Table 24.

Table 24 – Definition of Sub-MMS Services

Name of service		Object class	Note
1- confirmed services:	Status	vmd	note 1
	Identify	vmd	note 1
	Get Name List	vmd	note 1
	Delete-domain	dom	note 1
	Initiate-download-sequence	dom	note 1
	Download-segment	dom	note 1
	Terminate-download-sequence	dom	note 1
	Initiate-upload-sequence	dom	note 1
	Upload-segment	dom	note 1
	Terminate-upload-sequence	dom	note 1
	Get Domain Attributes	dom	note 1
	Create-program-invocation	pi	note 1
	Delete-program-invocation	pi	note 1
	Start	pi	note 1
	Stop	pi	note 1
	Resume	pi	note 1
	Reset	pi	note 1
	Kill	pi	note 1
	Get Program Invocation Attributes	pi	note 1
	Read	var/var-list	note 1
	Write	var/var-list	note 1
	Get Variable Access Attributes	var	note 1
	Define-variable-list	var-list	note 1
	Delete-variable-list	var-list	note 1
	Get Variable List Attributes	var-list	note 1
	Get-alarm-summary	event	note 1
	Acknowledge-event-notification	event	note 1
	Alter-event-condition-monitoring	event	note 1
	Get Event Condition Attributes	event	note 1
	Initiate	association	note 2
	Conclude	association	note 1
	Generic-Init-Download	OD	note 1
	Generic-Download	OD	note 1
	Generic-terminate-Download	OD	note 1
	Get OD Header/Data Type Attributes	OD	note 1
2-Unconfirmed services:	Unsolicited-Status	Vmd	note 1
	Information-Report	Var	note 1
	Event-Notification	Event	note 1
	Abort	association	note 2
	Reject	PDU	note 3
NOTE 1 service negotiable via the Initiate service. NOTE 2 service compulsory if the equipment supports the negotiable connection. NOTE 3 service compulsory for all the equipment supporting the indications of confirmed services (rejection of confirmed services requests).			

5.2.3.1.7 Definition of confirmed-service-req

The structure of a service request includes:

- 1) the service label ([X] = service code),
- 2) the service parameters (called "req-serv"),
- 3) the parameter extensions ensured by the CSs (called "req-detail").

For reasons of convenience parameters 2 and 3 are described apart.

NOTE The requests and responses of confirmed services are identified by the same service code. The requests of unconfirmed services are identified by service codes which are distinct from those of the confirmed services.

```

ConfirmedServiceReq:: =
--DOMAIN services--
    initiate-download-sequence
        req-serv
        req-detail
    download-segment
        req-serv
        req-detail
    terminate-download-sequence
        req-serv
        req-detail
    initiate-upload-sequence
        req-serv
        req-detail
    upload-segment
        req-serv
        req-detail
    terminate-upload-sequence
        req-serv
        req-detail
    delete-domain
        req-serv
        req-detail
--VMD services
    status
        req-serv
        req-detail
    identify
        req-serv
        req-detail
--PROGRAM INVOCATION services--
    create-program-invocation
        req-serv
        req-detail
    delete-program-invocation
        req-serv
        req-detail
    start
        req-serv
        req-detail
    stop
        req-serv
        req-detail

    resume
        req-serv
        req-detail
    reset
        req-serv
        req-detail
    kill
        req-serv
        req-detail
--VARIABLE /VARIABLE-LIST services--
    read
        req-serv
        req-detail
    write
        req-serv
        req-detail
    define-variable-list
        req-serv
        req-detail
    delete-variable-list
        req-serv
        req-detail
--EVENT services--
    get-alarm-summary
        req-serv
        req-detail
    acknowledge event notification
        req-serv
        req-detail
    alter-event-cond-monitor
        req-serv
        req-detail

```

```

CHOICE{
[6] IMPLICIT SEQUENCE{
    [0] IMPLICIT InitiateDownloadSequenceRequest,
    [1] IMPLICIT CSInitiateDownloadSequenceRequest},
[7] IMPLICIT SEQUENCE{
    [0] DownloadSegmentRequest,
    [1] IMPLICIT CSDownloadSegmentRequest},
[8] IMPLICIT SEQUENCE{
    [0] IMPLICIT TerminateDownloadSequenceRequest,
    [1] IMPLICIT CSTerminateDownloadSequenceRequest},
[9] IMPLICIT SEQUENCE{
    [0] InitiateUploadSequenceRequest,
    [1] IMPLICIT CSInitiateUploadSequenceRequest},
[10] IMPLICITSEQUENCE{
    [0] UploadSegmentRequest,
    [1] IMPLICIT CSUploadSegmentRequest},
[11] IMPLICITSEQUENCE{
    [0] TerminateUploadSequenceRequest,
    [1] IMPLICIT CSTerminateUploadSequenceRequest},
[12] IMPLICITSEQUENCE{
    [0] DeleteDomainRequest,
    [1] IMPLICIT CSDeleteDomainRequest},

[15] IMPLICITSEQUENCE{
    [0] IMPLICIT StatusRequest,
    [1] IMPLICIT CSStatusRequest},
[16] IMPLICIT SEQUENCE{
    [0] IMPLICIT IdentifyRequest,
    [1] IMPLICIT CSIdentifyRequest},

[20] IMPLICITSEQUENCE{
    [0] IMPLICIT CreateProgramInvocationRequest,
    [1] IMPLICIT CSCreateProgramInvocationRequest},
[21] IMPLICITSEQUENCE{
    [0] DeleteProgramInvocationRequest,
    [1] IMPLICIT CSDeleteProgramInvocationRequest},
[22] IMPLICITSEQUENCE{
    [0] IMPLICIT StartRequest,
    [1] IMPLICIT CSStartRequest},
[23] IMPLICITSEQUENCE{
    [0] StopRequest,
    [1] IMPLICIT CSStopRequest},

[24] IMPLICITSEQUENCE{
    [0] IMPLICIT ResumeRequest,
    [1] IMPLICIT CSResumeRequest},
[25] IMPLICITSEQUENCE{
    [0] ResetRequest,
    [1] IMPLICIT CSResetRequest},
[26] IMPLICITSEQUENCE{
    [0] KillRequest,
    [1] IMPLICIT CSKillRequest},

[30] IMPLICITSEQUENCE{
    [0] ReadRequest,
    [1] IMPLICIT CSReadRequest},
[31] IMPLICITSEQUENCE{
    [0] IMPLICIT WriteRequest,
    [1] IMPLICIT CSWriteRequest},
[35] IMPLICITSEQUENCE{
    [0] IMPLICIT DefineVariableListRequest,
    [1] IMPLICIT CSDefineVariableListRequest},
[36] IMPLICITSEQUENCE{
    [0] DeleteVariableListRequest,
    [1] IMPLICIT CSDeleteVariableListRequest},

[39] IMPLICITSEQUENCE{
    [0] GetAlarmSummaryRequest,
    [1] IMPLICIT CSGetAlarmSummaryRequest},
[40] IMPLICITSEQUENCE{
    [0] IMPLICIT AcknowledgeEventNotificationRequest,
    [1] IMPLICIT CSAcknowledgeEventNotificationRequest},
[41] IMPLICITSEQUENCE{
    [0] IMPLICIT AlterEventConditionMonitoringRequest,
    [1] IMPLICIT CSAAlterEventConditionMonitoringRequest},

```

```

--Get Attributes services--
  get-name-list          [45] IMPLICITSEQUENCE{
    req-serv              [0] IMPLICIT GetNameListRequest,
    req-detail            [1] IMPLICIT CSGetNameListRequest},
  get-domain-attributes [46] IMPLICITSEQUENCE{
    req-serv              [0] GetDomainAttributesRequest,
    req-detail            [1] IMPLICIT CSGetDomainAttributesRequest},
  get-program-invocation-attributes [47] IMPLICITSEQUENCE{
    req-serv              [0] GetProgramInvocationAttributesRequest,
    req-detail            [1] IMPLICIT CSGetProgramInvocationAttributesRequest},
  get-variable-access-attributes [48] IMPLICITSEQUENCE{
    req-serv              [0] GetVariableAccessAttributesRequest,
    req-detail            [1] IMPLICIT CSGetVariablesAccessAttributesRequest},
  get-variable-list-attributes [49] IMPLICITSEQUENCE{
    req-serv              [0] GetVariableListAttributesRequest,
    req-detail            [1] IMPLICIT CSGetVariableListAttributesRequest},
  get-event-condition-attributes [50] IMPLICITSEQUENCE{
    req-serv              [0] GetEventConditionAttributesRequest,
    req-detail            [1] IMPLICIT CSGetEventConditionAttributesRequest},
--OD services--
  generic-init-download [55] IMPLICITSEQUENCE{
    req-serv              [0] IMPLICIT GenericInitDownloadRequest,
    req-detail            [1] IMPLICIT CSGenericInitDownloadRequest},
  generic-download      [56] IMPLICITSEQUENCE{
    req-serv              [0] IMPLICIT GenericDownloadRequest,
    req-detail            [1] IMPLICIT CSGenericDownloadRequest},
  generic-terminate-download [57] IMPLICITSEQUENCE{
    req-serv              [0] IMPLICIT GenericTerminateDownloadRequest,
    req-detail            [1] IMPLICIT CSGenericTerminateDownloadRequest},
  get-OD-header/datatype-attributes [58] IMPLICITSEQUENCE{
    req-serv              [0] IMPLICIT GetODHeaderDataTypeRequest,
    req-detail            [1] IMPLICIT CSGetODHeaderDataTypeRequest},
}

```

The ConfirmedServiceReq includes:

- 1) [X] identifies the service,
- 2) req-serv references the production corresponding to the selected service,
- 3) req-detail references the production of the Companion Standards (CS) corresponding to the selected service.

5.2.3.1.8 Definition of unconfirmed-service

The unconfirmed services are transmitted spontaneously. The reason for their transmission is not defined by this document (local issue).

The choice of the channel(s) via which a unconfirmed service is transmitted is also a local issue. Consequently, a notification can take place in point to point and/or in multipoint and/or in broadcasting.

The structure of the service includes:

- 1) the service label ([X] = service code),
- 2) the parameter services (called "req-serv"),
- 3) the parameter extensions by the CS (called "req-detail").

For reasons of convenience, parameters 3 and 4 are described separately.

NOTE The requests of unconfirmed services are identified by service codes distinct from those of the confirmed services.

```

UnconfirmedService ::= CHOICE{
--VMD service--
    unsolicited-status          [3] IMPLICIT SEQUENCE{
        req-serv                [0] IMPLICIT UnsolicitedStatus,
        req-detail              [1] IMPLICIT CSUnsolicitedStatus},
--EVENT service:
    event-notification          [4] IMPLICIT SEQUENCE{
        req-serv                [0] IMPLICIT EventNotification,
        req-detail              [1] IMPLICIT CSEventNotification},
--VARIABLE service--
    information-report          [5] IMPLICIT SEQUENCE{
        req-serv                [0] IMPLICIT InformationReport,
        req-detail              [1] IMPLICIT CSInformationReport},
}

```

The unconfirmedService includes:

- 1) [X] identifies the service,
- 2) req-serv reference of the production corresponding to the selected service,
- 3) req-detail references the production of the Companion Standards (CS) corresponding to the selected service.

5.2.3.1.9 Definition of confirmed-service-req

The structure of the service includes:

- 1) the service label([X] = service code),
- 2) the service parameters (called "rsp-serv"),
- 3) the parameter extensions made by the CS (called "rsp-detail").

For reasons of convenience, parameters 2 and 3 are described elsewhere.

NOTE The requests and responses of confirmed services are identified by the same service code. The requests of unconfirmed services are identified by service codes which are distinct from those of the confirmed services.

```

ConfirmedServiceResp ::= CHOICE{
--DOMAIN services--
    initiate-download-sequence [6] IMPLICIT SEQUENCE{
        rsp-serv                [0] IMPLICIT InitiateDownloadSequenceResponse,
        rsp-detail              [1] IMPLICIT CSInitiateDownloadSequenceResponse},
    download-segment           [7] IMPLICIT SEQUENCE{
        rsp-serv                [0] IMPLICIT DownloadSegmentResponse,
        rsp-detail              [1] IMPLICIT CSDownloadSegmentResponse},
    terminate-download-sequence [8] IMPLICIT SEQUENCE{
        rsp-serv                [0] IMPLICIT TerminateDownloadSequenceResponse,
        req-detail              [1] IMPLICIT CSTerminateDownloadSequenceResponse},
    initiate-upload-sequence    [9] IMPLICIT SEQUENCE{
        rsp-serv                [0] IMPLICIT InitiateUploadSequenceResponse,
        rsp-detail              [1] IMPLICIT CSInitiateUploadSequenceResponse},
    upload-segment             [10] IMPLICIT SEQUENCE{
        rsp-serv                [0] IMPLICIT UploadSegmentResponse,
        rsp-detail              [1] IMPLICIT CSUploadSegmentResponse},
    terminate-upload-sequence   [11] IMPLICIT SEQUENCE{
        rsp-serv                [0] IMPLICIT TerminateUploadSequenceResponse,
        rsp-detail              [1] IMPLICIT CSTerminateUploadSequenceResponse},
    delete-domain              [12] IMPLICIT SEQUENCE{
        rsp-serv                [0] IMPLICIT DeleteDomainResponse,
        rsp-detail              [1] IMPLICIT CSDeleteDomainRspues},
--VMD services
    status                     [15] IMPLICIT SEQUENCE{
        rsp-serv                [0] IMPLICIT StatusResponse,
        rsp-detail              [1] IMPLICIT CSStatusResponse},
    identify                   [16] IMPLICIT SEQUENCE{
        rsp-serv                [0] IMPLICIT IdentifyResponse,
        rsp-detail              [1] IMPLICIT CSIdentifyResponse},
--PROGRAM INVOCATION services--
    create-program-invocation  [20] IMPLICIT SEQUENCE{
        rsp-serv                [0] IMPLICIT CreateProgramInvocationResponse,
        rsp-detail              [1] IMPLICIT CSCreateProgramInvocationResponse},
    delete-program-invocation [21] IMPLICIT SEQUENCE{
        rsp-serv                [0] IMPLICIT DeleteProgramInvocationResponse,
        rsp-detail              [1] IMPLICIT CSDeleteProgramInvocationResponse},
}

```

start	[22] IMPLICITSEQUENCE{
rsp-serv	[0] IMPLICIT StartResponse,
rsp-detail	[1] IMPLICIT CSStartResponse},
stop	[23] IMPLICITSEQUENCE{
rsp-serv	[0] IMPLICIT StopResponse,
rsp-detail	[1] IMPLICIT CSStopResponse},
resume	[24] IMPLICITSEQUENCE{
rsp-serv	[0] IMPLICIT ResumeResponse,
rsp-detail	[1] IMPLICIT CSResumeResponse},
reset	[25] IMPLICITSEQUENCE{
rsp-serv	[0] IMPLICIT ResetResponse,
rsp-detail	[1] IMPLICIT CSResetResponse},
kill	[26] IMPLICITSEQUENCE{
rsp-serv	[0] IMPLICIT KillResponse,
rsp-detail	[1] IMPLICIT CSKillResponse},
--VARIABLE/VARIABLE-LIST services--	
read	[30] IMPLICITSEQUENCE{
rsp-serv	[0] IMPLICIT ReadResponse,
rsp-detail	[1] IMPLICIT CSReadResponse},
write	[31] IMPLICITSEQUENCE{
rsp-serv	[0] IMPLICIT WriteResponse,
rsp-detail	[1] IMPLICIT CSWriteResponse},
define-variable-list	[35] IMPLICITSEQUENCE{
rsp-serv	[0] IMPLICIT DefineVariableListResponse,
rsp-detail	[1] IMPLICIT CSDefineVariableListResponse},
delete-variable-list	[36] IMPLICITSEQUENCE{
rsp-serv	[0] IMPLICIT DeleteVariableListResponse,
rsp-detail	[1] IMPLICIT CSDeleteVariableListResponse},
--EVENT services--	
get-alarm-summary	[39] IMPLICITSEQUENCE{
rsp-serv	[0] IMPLICIT GetAlarmSummaryResponse,
rsp-detail	[1] IMPLICIT CSGetAlarmSummaryResponse},
acknowledge-event-notification	[40] IMPLICITSEQUENCE{
rsp-serv	[0] IMPLICIT AcknowledgeEventNotificationResponse,
rsp-detail	[1] IMPLICIT CSAcknowledgeEventNotificationResponse},
alter-event-cond-monitor	[41] IMPLICITSEQUENCE{
rsp-serv	[0] IMPLICIT AlterEventConditionMonitoringResponse,
rsp-detail	[1] IMPLICIT CSAAlterEventConditionMonitoringResponse},
--Get Attributes services--	
get-name-list	[45] IMPLICITSEQUENCE{
req-serv	[0] GetNameListResponse,
req-detail	[1] IMPLICIT CSGetNameListResponse},
get-domain-attributes	[46] IMPLICITSEQUENCE{
req-serv	[0] IMPLICIT GetDomainAttributesResponse,
req-detail	[1] IMPLICIT CSGetDomainAttributesResponse},
get-program-invocation-attributes	[47] IMPLICITSEQUENCE{
req-serv	[0] IMPLICIT GetProgramInvocationAttributesResponse,
req-detail	[1] IMPLICIT CSGetProgramInvocationAttributesResponse},
get-variable-access-attributes	[48] IMPLICITSEQUENCE{
req-serv	[0] IMPLICIT GetVariableAccessAttributesResponse,
req-detail	[1] IMPLICIT CSGetVariablesAccessAttributesResponse},
get-variable-list-attributes	[49] IMPLICITSEQUENCE{
req-serv	[0] IMPLICIT GetVariableListAttributesResponse,
req-detail	[1] IMPLICIT CSGetVariableListAttributesResponse},
get-event-condition-attributes	[50] IMPLICITSEQUENCE{
req-serv	[0] IMPLICIT GetEventConditionAttributesResponse,
req-detail	[1] IMPLICIT CSGetEventConditionAttributesResponse},
--OD services--	
generic-init-download	[55] IMPLICITSEQUENCE{
req-serv	[0] IMPLICIT GenericInitDownloadRequest,
req-detail	[1] IMPLICIT CSGenericInitDownloadRequest},
generic-download	[56] IMPLICITSEQUENCE{
req-serv	[0] IMPLICIT GenericDownloadRequest,
req-detail	[1] IMPLICIT CSGenericDownloadRequest},
generic-terminate-download	[57] IMPLICITSEQUENCE{
req-serv	[0] IMPLICIT GenericTerminateDownloadRequest,
req-detail	[1] IMPLICIT CSGenericTerminateDownloadRequest},
get-OD-header/datatype-attributes	[58] IMPLICITSEQUENCE{
req-serv	[0] IMPLICIT GetODHeaderDataTypeRequest,
req-detail	[1] IMPLICIT CSGetODHeaderDataTypeRequest},
}	

ConfirmedServiceResp includes three parameters:

- 1) [X] identifies the service,
- 2) rsp-serv references the production corresponding to the selected service,

3) rsp-detail references the production of the CS corresponding to the selected service.

5.2.3.1.10 Definition of error-type

```
ErrorType ::= CHOICE{
--Choice of a sub-set of MMS standard errors. The CSs can lengthen the MMS standard error list.--
--MMS standard errors:
    vmd [0] IMPLICIT UNSIGNED-16{
        other (0)},
    application-reference [1] IMPLICIT UNSIGNED-16{
        other (0),
        application-unreachable (1),
        -- value (2) reserved for connection lost,
        application-reference-invalid (3),
        context-unsupported (4)},
    definition [2] IMPLICIT UNSIGNED-16 {
        other (0),
        object-undefined (1),
        --value (2) reserved for invalid-address --
        type-unsupported (3),
        type-inconsistent (4),
        object exist (5),
        object-attribute-inconsistent (6)},
    resource [3] IMPLICIT UNSIGNED-16 {
        other (0),
        memory-unavailable (1),
        --value (2) reserved for processor resource unavailable,--
        --value (3) reserved for mass storage unavailable,--
        capability-unavailable (4),
        capability-unknown (5)},
    service [4] IMPLICIT UNSIGNED-16 {
        other (0),
        primitive-out-of-sequence (1),
        object-state-conflict (2)+256*valeur "pi-state",
        --value (3) reserved for further definition,--
        --value (4) reserved for continuation invalid,--
        object-constraint-conflict (5)+256*valeur "pi-state",
        service-preempt [5] IMPLICIT UNSIGNED-16 {
            other (0),
            timeout (1),
            --value (2) reserved for deadlock,--
            --value (3) reserved for cancel--},
        --choice-value [6] reserved for time resolution,--
        access [7] IMPLICIT UNSIGNED-16 {
            other (0),
            object-access-unsupported (1),
            object-non-existent (2),
            object-access-denied (3),
            object-invalidated (4)},
        initiate [8] IMPLICIT UNSIGNED-16 {
            other (0),
            --values (1) and (2) are reserved for further definition--
            max serv outstanding calling insufficient (3),
            max serv outstanding called insufficient (4),
            service CBB insufficient (5),
            parameter CBB insufficient (6),
            nesting level insufficient (7) },
        conclude [9] IMPLICIT UNSIGNED-16 {
            other (0),
            further-communication-required (1)},
        --choice value [10] reserved for cancel,--
        --choice value [11] reserved for file,--
        other [12] IMPLICIT UNSIGNED-16 {
            --values reserved for local matter--
        }
}
```

In case of failure of the Start, Stop, Resume and Reset services, the Error-Type also contains information on the "class error" and "code error" the value of "pi-state". In case of failure of the other services, the Error-Type should contain information on "class error" and "code error" and the value of the "pi-state" equals (0).

NOTE 1 [x] represents the error class; this is identical to the equivalent MMSs.

NOTE 2 (y) represents the error code; this is identical to the equivalent MMS.

NOTE 3 The authorized values of the "pi-state" are those defined.

5.2.3.1.11 Definitions of the data types

Specifications of the types of data exchanged (data = content of a variable or "ErrorType") allows description of the exact composition of data, indicating the type and length.

```

TypeDataSpecification ::= CHOICE{
--The CSs and the Sub-MMS users can increase this list.--
--Each type extension shall be identified by [X] IMPLICIT "Y". The values of "X" varying from 15 to 25 are reserved
for types defined by the Companion Standards. The values of "X" varying from 26 to 127 are reserved for types
defined by the users. "Y" shall be defined using the EN 50170 - Volume 3 ASN.1 notation (see appendix A)--
--DATA-TYPE--
--constructed data type --
    array-type                                [0] IMPLICIT SEQUENCE{
        number-of-elements                    UNSIGNED-8,
        element-type                          TypeDataSpecification
    }
--recommended: nest=1--},
    structure-type                            [1] IMPLICIT SEQUENCE OF{
        TypeDataSpecification
    }
--recommended: nest=1--},
--Simple data type--
    Boolean-type                             [2] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 represents the length; the length is the number of bits used to encode the Boolean value;
authorized length values = 8.
    integer-type                             [3] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 represents the length; length is the number of bits used to encode the integer value; authorized
length values = multiple of 8.--
    unsigned-type                             [4] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 represents the length; length is the number of bits used to encode the unsigned value; authorized
length values = multiples of 8.--
    bitstring-type                           [5] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 represents the length; length is the number of bits used to encode the bitstring value --
    visiblestring-type                       [6] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 represents the length; length is the number of octets used to encode the visiblestring value.--
    octetstring-type                         [7] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 represents the length; length is the number of octets used to encode the octetstring value.--
    time-difference                         [8] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 represents the length; length is the number of octets used to encode the time-difference value;
authorized length value = 4.--
    time-of-day-type                        [9] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 represents the length; length is the number of octets used to encode the time of day value;
authorized length value = 6.--
    single-floating-type                     [10] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 represents the length; length is the number of octets used to encode the single floating value;
authorized length value= 4.--
    double-floating-type                    [11] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 represents the length; length is the number of octets used to encode the double floating value;
authorized length value = 8 --
    bcd-type                                [12] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 represents the length; length is the number of digits--
    Boolean-array-type                      [13] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 represents the length; length is the number of bits used to encode the value of the Boolean array.--
--ERROR-TYPE--
    error-type                              [14] IMPLICIT UNSIGNED-8
--UNSIGNED-8 represents the length; length is the number of octets used to encode the error value; authorized
length value = 3.--
}

```

5.2.3.1.12 Definitions of the data values (without length or type)

- The CSs and the Sub-MMS users can increase this list of Data Values by using the ASN.1 notation.
- The following values, resulting from encoding of the types of ASN.1 data and completed by other types of data judged useful, correspond to data whose type and length are either implicitly known or specified by TypeDataSpecification, or explicitly specified in a service. The type and length not included in the encoding, however the value subsists. In the framework of variable management and events, the number of octets used for encoding the value shall correspond with that specified by TypeDataSpecification. These data are used for example in the Read, Write, Information Report services, etc.--

```

DATA VALUE ::= CHOICE{
--Structured data values--
Array ::= OCTETSTRING SIZE (n)
--OCTETSTRING-SIZE (n) is composed of the juxtaposition of the simple data values composing it--
}

```

5.2.3.1.13 Complementary definitions

ListOfObjectType ::= SEQUENCE OF TypeDataSpecification

ListOfObjectValue ::= OCTET STRING

--it is the juxtaposition of the values of the variables and/or error types encoded in compliance with "Data Value" or Error type

ListOfErrorType ::= OCTET STRING

--contains the juxtaposition of several ErrorType values encoded in compliance with Error Type (See 5.2.3.1.10).

```
{
ObjectClass ::= UNSIGNED-8
  Null          (0),
  Domain        (1),
  PI             (2),
  Variable      (3),
  Variable-List (4),
  Event         (5),
  OD Header     (6),
  Data Type     (7).
}
```

5.2.4 VMD ASE Protocol

5.2.4.1 Environment management protocol

5.2.4.1.1 General

The association management protocol defines the PDUs necessary for execution of the following services:

- 1) Initiate,
- 2) Conclude,
- 3) Abort,
- 4) Reject.

5.2.4.1.2 Definition of the confirmed initiate service protocol

NOTE Init Detail Request and Init Detail Response are defined in the SUB-MMS-CORE, module,

```
InitiateReqPDU ::= SEQUENCE {
  quality-of-service-calling          QualityOfServiceCalling
  extension-calling                   ExtensionCalling,
  proposed-max-serv-outstanding-calling INTEGER-16,
  proposed-max-serv-outstanding-called INTEGER-16,
  proposed-data-structure-nesting    INTEGER-8,
  --recommended nest=1; shall be ≥ 1--
  init-detail-request                [0] IMPLICIT InitDetailRequest
}
```

```
InitiateRespPDU ::= SEQUENCE {
  quality-of-service-called          QualityOfServiceCalled,
  extension-called                   ExtensionCalled,
  negotiated-max-serv-outstanding-calling INTEGER-16,
  negotiated-max-serv-outstanding-called INTEGER-16,
  negotiated-data-structure-nesting  INTEGER-8,
  --recommended nest=1; shall be ≥ 1--
  init-detail-response              [0] IMPLICIT InitDetailResponse
  --see 10.2.8--
}
```

```
QualityOfServiceCalling ::= BITSTRING-8 {
  quality-of-service (0),
  -- quality of service 1 (value of the bit = 1)--
  --quality of service 2 (value of the bit = 0)--
}
```

ExtensionCalling ::= OCTETSTRING

- For quality 1 the content shall be empty--
- For quality 2 the content shall be AccessControl--
- For service quality extensions, the format of this parameter shall be specified.

QualityOfServiceCalled ::= BITSTRING-8 {
 quality-of-service (0)
 -- quality of service 1 (value of the bit = 1): no information concerning this quality shall be included in
 ExtensionCalling.--
 --quality of service 2 (value of the bit = 0): the information concerning this quality shall be of the AccessControl
 type; this information shall be present in ExtensionCalling and be included in the first row (first two octets of
 OCTETSTRING content)--
}

ExtensionCalled: : = OCTETSTRING
 }

AccessControl : : = BITSTRING-16 {
 password bit 8 (0),
 password bit 7 (1),
 password bit 6 (2),
 password bit 5 (3),
 password bit 4 (4),
 password bit 3 (5),
 password bit 2 (6),
 password bit 1 (7),

 access group 8 (8),
 access group 7 (9),
 access group 6 (10),
 access group 5 (11),
 access group 4 (12),
 access group 3 (13),
 access group 2 (14),
 access group 1 (15) }

InitiateErrorPDU::= ErrorType

ParameterCBBSupportOption::= BITSTRING{
 --true value=parameter supported. The CSs can extend this list--
 --types of data
 arr (array) (0),--bit No. 0
 str (structure) (1),--bit No. 1
 vlis (variable-list) (2),--bit No. 2
 valt (partial access) (3),--bit No. 3
}

ServiceSupportOption::= BITSTRING {
 --A service is supported when its bit takes the true value (1). The CSs can lengthen this list by adding new
 services.
 a) the sub-set of bits No. 0 to No. 39 indicates the support of the indications of services; this sub-set shall be
 present in the BITSTRING.
 b)the sub-set of bits No. 40 to No. 79 indicates the support of the requests for services and the bits extending
 from No. 40 to No. 81 indicate respectively the support of the name for the indications of services and the
 requests for services; this sub-set is optional for the request and conditional for the response .
 The length of the bitstring can thus take the value 39, or the value 81--

--Bits No. 0 to 32 are mandatory
--vmd management services supported
 status.ind (0), --bit No. 0
 unsolicited status.ind (unconfirmed) (1), --bit No. 1
 identify.ind (2), --bit No. 2
-- domain management services supported
 delete domain.ind (3), --bit No. 3
 initiate download sequence.ind (4), --bit No. 4
 download segment.ind (5), --bit No. 5
 terminate download sequence.ind (6), --bit No. 6
 initiate upload sequence.ind (7), --bit No. 7
 upload segment.ind (8), --bit No. 8
 terminate upload sequence.ind (9), --bit No. 9
-- program invocations management services supported
 create program invocation.ind (10),--bit No. 10
 delete program invocation.ind (11),--bit No. 11
 start.ind (12),--bit No. 12
 stop.ind (13),--bit No. 13
 resume.ind (14),--bit No. 14
 reset.ind (15),--bit No. 15
 kill.ind (16),--bit No. 16
-- variable management services supported
 read.ind (17),--bit No. 17
 write.ind (18),--bit No. 18
 information report.ind (unconfirmed) (19),--bit No. 19
 define variable-list.ind (20),--bit No. 20
 delete variable-list.ind (21),--bit No. 21

<u>-- event management services supported</u>	
event notification.ind (unconfirmed)	(22), --bit No. 22
alter event condition monitoring.ind	(23), --bit No. 23
acknowledge event notification.ind	(24), --bit No. 24
get alarm summary.ind	(25), --bit No. 25
<u>-- context management services supported</u>	
conclude.ind	(26), --bit No. 26
<u>-- attributes management services supported</u>	
get name list.ind	(27), --bit No. 27
get domain attributes.ind	(28), --bit No. 28
get program invocation attributes.ind	(29), --bit No. 29
get variable access attributes.ind	(30), --bit No. 30
get variable-list attributes.ind	(31), --bit No. 31
get event condition attributes.ind	(32), --bit No. 32
<u>-- OD services</u>	
generic-init-download.ind	(33), --bit No. 33
generic-download.ind	(34), --bit No. 34
generic-terminate-download.ind	(35), --bit No. 35
get OD Header/Data Type.ind	(36), --bit No. 36
<u>--Bits No. 37 to 39 are reserved for extensions of services</u>	
<u>--The set of bits from 40 to 80 is optional for the request and conditional for the response</u>	
<u>--vmd management services supported</u>	
status.req	(40), --bit No. 40
unsolicited status.req (unconfirmed)	(41), --bit No. 41
identify.req	(42), --bit No. 42
<u>-- domain management services supported</u>	
delete domain.req	(43), --bit No. 43
initiate download sequence.req	(44), --bit No. 44
download segment.req	(45), --bit No. 45
terminate download sequence.req	(46), --bit No. 46
initiate upload sequence.req	(47), --bit No. 47
upload segment.req	(48), --bit No. 48
terminate upload sequence.req	(49), --bit No. 49
<u>-- program invocation management services supported</u>	
create program invocation.req	(50), --bit No. 50
delete program invocation.req	(51), --bit No. 51
start.req	(52), --bit No. 52
stop.req	(53), --bit No. 53
resume.req	(54), --bit No. 54
reset.req	(55), --bit No. 55
kill.req	(56), --bit No. 56
<u>-- variable management services supported</u>	
read.req	(57), --bit No. 57
write.req	(58), --bit No. 58
information report.req (unconfirmed)	(59), --bit No. 59
define variable-list.req	(60), --bit No. 60
delete variable-list.req	(61), --bit No. 61
<u>-- event management services supported</u>	
event notification.req (unconfirmed)	(62), --bit No. 62
alter event condition monitoring.req	(63), --bit No. 63
acknowledge event notification.req	(64), --bit No. 64
get alarm summary.req	(65), --bit No. 65
<u>-- context management service supported</u>	
conclude.req	(66), --bit No. 66
<u>-- attributes management services supported</u>	
get name list.req	(67), --bit No. 67
get domain attributes.req	(68), --bit No. 68
get program invocation attributes.req	(69), --bit No. 69
get variable access attributes.req	(70), --bit No. 70
get variable-list attributes.req	(71), --bit No. 71
get event condition attributes.req	(72), --bit No. 72
<u>-- OD services</u>	
generic-init-download.ind	(73), --bit No. 73
generic-download.ind	(74), --bit No. 74
generic-terminate-download.ind	(75), --bit No. 75
get OD Header/Data Type.ind	(76), --bit No. 76
<u>--Bits No. 77 to 79 are reserved for extensions of services</u>	
<u>--Type of identification supported</u>	
name support for service.ind	(80), --bit No. 80
name support for service.req	(81), --bit No. 81

}

5.2.4.1.3 Definition of confirmed conclude service protocol

ConcludeReqPDU ::= NULL.

ConcludeRespPDU ::= NULL

ConcludeErrorPDU ::= ErrorType

5.2.4.1.4 Definition of the unconfirmed abort service protocol

The Abort service is mapped directly on the A-Abort service of the lower layer ensuring the connections.

5.2.4.1.5 Definition of the unconfirmed reject service protocol

The reject service is described by Reject PDU.

```
RejectPDU ::= SEQUENCE{
    original-invocation-Id          UNSIGNED-32,
    --not significant if the reject concerns Conclude--
    reject-reason                   RejectReason}

RejectReason ::= CHOICE {
    --The CSs can lengthen the following list --
    confirmed-request-PDU          [0] IMPLICIT UNSIGNED-8{
        other                      (0),
        unrecognized-service       (1),
        invalid invocation Id      (3),
        invalid-argument           (4),
        max-serv-outstanding-exceeded (6),
        max-recursion-exceeded     (8),
        value-out-of-range         (9)},
    pdu-error                      [4] IMPLICIT UNSIGNED-8{
        unknown-type-pdu          (0),
        invalid-pdu               (1),
        illegal-mapping           (2)},
    conclude-request-PDU          [8] IMPLICIT UNSIGNED-8{
        other                      (9),
        invalid-argument           (1)},
    }
}
```

5.2.4.2 Protocol concerning the vmd management

5.2.4.2.1 General

This section defines the protocols of the following services:

- 1) Status,
- 2) UnsolicitedStatus,
- 3) Identify,
- 4) Get Name List.

5.2.4.2.2 Definition of the confirmed status service protocol

– request parameters.

StatusRequest ::= BOOLEAN --true for extended derivation

– response parameters.

```

StatusResponse ::= SEQUENCE{
    vmd-logical-status          UNSIGNED-8{
        state-change-allowed    (0),
        no-state-change-allowed (1),
        others                   (2 to 5),
        --reserved for complementary states. These states shall be explained in the PICS.
        These complementary states allow ensuring the compatibility with MMS, FMS. Values 2 to 3 are reserved to
        ensure compatibility of Sub-MMS with MMS. Value 2 is reserved for the limited service permitted state. Value 3
        is reserved for the support service allowed state. Values 4 to 5 are reserved to ensure the compatibility of Sub-
        MMS with FMS. Value 4 is reserved for the state "loading of objects without consequences for the users". Value
        5 is reserved for the state "loading of objects with consequences for the users".
        (6 to n)
    },
    vmd-physical-status          UNSIGNED-8{
        operational              (0),
        partially-operational    (1),
        inoperable               (2),
        needs-commissioning      (3) },
    local-detail                 BITSTRING SIZE( 0..128)
    --the content of BIT STRING is empty if the local detail is not used --
}

```

- 1) "State change allowed" (value 0 for UNSIGNED-8):
all the services of Sub-MMS are authorized.
- 2) "No state change allowed" (value 1 for UNSIGNED-8):
the following services are authorized.

Initiate,
 Conclude,
 Abort,
 Status,
 Identify.
 Read
 Get Alarm Summary
 Get Name List
 Get Domain Attributes
 Get Program Invocation Attributes
 Get Variable Access Attributes
 Get Variable List Attributes
 Get Event Condition Attributes

5.2.4.2.3 Definition of the unconfirmed unsolicited status service protocol

- service parameters.

UnsolicitedStatus ::= StatusResponse

5.2.4.2.4 Definition of the confirmed identify service protocol

- request parameters.

IdentifyRequest :: = NULL

- response parameters.

```

IdentifyResponse ::= SEQUENCE {
    vendor's name  VISIBLESTRING,
    model name     VISIBLESTRING,
    revision       VISIBLESTRING
}

```

5.2.4.2.5 Definition of the confirmed get name list service protocol

– request parameters.

```

GetNameListRequest ::= SEQUENCE {
    object class          OBJECTCLASS {
        --values authorized
        null              (0),
        domain            (1),
        pi                (2),
        variable          (3),
        variable-list     (4),
        event             (5),
        ODHeader          (6),
        Datatype          (7).
    }
    continue after      identification
}
    
```

– response parameters.

```

GetNameListResponse ::= CHOICE {
    list-of-ID          SEQUENCE {
        --selected if the object class value is different from 0
        list-of-identification ListOfObjectId,
        -- if an object supports the double identification (name and index) only the index will be transmitted--
        more follows          BOOLEAN },
    list-of-description  SEQUENCE {
        --selected if the value of the object class is equal to 0
        object description    CHOICE {
            gives the description of the object
            domain-description GetDomainAttributesResponse
            pi-description     GetProgramInvocationAttributesResponse
            variable-description GetVariableAccessAttributesResponse
            variable-list-description GetVariableListAttributesResponse
            event-description  GetEventConditionAttributesResponse
            od-header-description GetODHeader/DataTypeAttributesResponse
            data-type-description GetODHeader/DataTypeAttributesResponse
        }
        more follows          BOOLEAN
    }
}
    
```

5.2.5 Domain ASE protocol

5.2.5.1 General

This section defines the protocols of the following services:

- 1) DeleteDomain,
- 2) InitiateDownloadSequence,
- 3) DownloadSegment,
- 4) TerminateDownloadSequence,
- 5) InitiateUploadSequence,
- 6) UploadSegment,
- 7) TerminateUploadSequence,
- 8) Get Domain Attributes.

NOTE The status of a Domain can implicitly take the values: "predefined/non-existent", "ready" and "in-use".

– a domain can preexist or can be created by a Sub-MMS service.

– a preexistent domain can either come up with its total data or partial data; the rest of its data can be loaded by the Sub-MMS services.

– the Delete Domain service can be executed locally.

5.2.5.2 Definition of the confirmed delete domain service protocol

– Request parameters.

DeleteDomainRequest: = Identification.

– Response parameters.

DeleteDomainResponse::= NULL

5.2.5.3 Definition of the confirmed initiate download sequence service protocol

– Request parameters.

```
InitDownloadSequenceRequest::= SEQUENCE{
    domain-identification          Identification,
    list-of-capabilities           SEQUENCE OF OCTETSTRING
-- the content can be blank (caution: the SEQUENCE OF encoding continues)
    sharable                      BOOLEAN--true for sharable
    domain-access-protection      DomainAccessProtection
}
```

NOTE The "domain-access-protection" parameter is assigned to the Domain object created by the service. It characterizes the level of protection of the object with respect to the CLIENTS.

DomainAccessProtection::= BITSTRING{

--For a service 1 quality, the content of BITSTRING shall be empty.

For a service 2 quality of the content the BITSTRING shall correspond to the following definition--

--password--

password-bit 8	(0),
password-bit 7	(1),
password-bit 6	(2),
password-bit 5	(3),
password-bit 4	(4),
password-bit 3	(5),
password-bit 2	(6),
password-bit 1	(7),

--access group--

access-group 8	(8),
access-group 7	(9),
access-group 6	(10),
access-group 5	(11),
access-group 4	(12),
access-group 3	(13),
access-group 2	(14),
access-group 1	(15),

--access object protection--

--value (16) is reserved--

Ug	(17),
----	-------

--If the value of the bit = 1 and the client belongs to one of the groups specified in the "access group" attribute, then the domain can be attached to a PI by the Create PI service--

Wg	(18),
----	-------

--If the value of the bit = 1 and the client belongs to one of the groups specified in the "access group" attribute, the domain can be downloaded/deleted by the Initiate Download Sequence, Download Segment, Terminate Download Sequence and Delete Domain services--

Rg	(19),
----	-------

--If the value of the bit = 1 and the client belongs to one of the groups specified in the "access group" attribute, then the domain can be uploaded by the Initiate Upload Sequence, Upload Segment and Terminate Upload Sequence services--

--value (20) is reserved--

U	(21),
---	-------

--If the value of the bit = 1 and the client has the required password identical to the attribute "password" then the domain can be linked to a PI by the Create PI service--

W	(22),
---	-------

--If the value of the bit = 1 and the client has given the required password identical to the "password" attribute, the domain can be downloaded by the Initiate Download Sequence, Download Segment, Terminate Download Sequence and Delete Domain services--

R	(23),
---	-------

--If the value of the bit = 1 and the client has given the required password identical to the attribute "password", the domain can be uploaded by the Initiate Upload Sequence, Upload Segment and Terminate Upload Sequence services--

```
--The values (24), (25), (26), (27), (28) are reserved --
Ua                                     (29),
--If the value of the bit = 1, all the clients can link the domain to a PI by the Create PI service--
Wa                                     (30),
--If the value of the bit = 1, all the clients can download/delete the domain by the Initiate Download Sequence,
Download Segment, Terminate Download Sequence and Delete Domain services--
Ra                                     (31)
--If the value of the bit = 1 all the clients can then upload the domain by the Initiate Upload Sequence, Upload
Segment and Terminate Upload Sequence services-
}
```

– Response parameters.

InitDownloadSequenceResponse ::= NULL
Definition of the confirmed download segment service protocol

– Request parameters

DownloadSegmentRequest ::= Identification.

– Response parameters

```
DownloadSegmentResponse ::= SEQUENCE{
    load-data                               OCTETSTRING,
    more-follow                             BOOLEAN
    --If the value is true, this means that there are other segments to be downloaded--
}
```

Definition of the confirmed terminate download sequence service protocol

– Request parameters.

```
TermDownloadSequenceRequest ::= SEQUENCE{
    domain-Id                               Identification,
    discard                                 Discard
    --the content of Discard is empty if the loading of the domain is successful, otherwise it contains ErrorType
    (caution: the length encoding continues as long as the content of the Discard is empty) see 5.2.3.1.11.
}
```

– Response parameters.

TermDownloadSequenceResponse ::= NULL

Definition of the confirmed initiate upload sequence service protocol

– Request parameters

InitUploadSequenceRequest ::= Identification.

– Response parameters

```
InitUploadSequenceResponse ::= SEQUENCE{
    ulsm Index                               Index,
    --Index is of the UNSIGNED-16 type. if the index is significant (the value of the UNSIGNED-16 is ≤32K-1) the
    saving should be continued with the ulsm index; if, on the contrary, the index is non-significant (the value of the
    UNSIGNED-16 is 32K) the saving shall continue with the domain identifier(the Name or Index of the domain). The
    resolution of the non ambiguity between domain Index and ULSM Index is at the server's charge.--
    list-of-capabilities                     SEQUENCE OF OCTETSTRING }
    --the content of the list of capabilities can be empty (caution: the SEQUENCE OF encoding continues)--
```

5.2.5.4 Definition of the confirmed upload segment service protocol

– Request parameters

UploadSegmentRequest ::= Identification

--See the above constraints of the Initiate Upload Sequence service on the significance of the "Identification".

"Identification" corresponds either to the name or to the domain Index or to the ULSM Index depending on the context.--

– Response parameters

```
UploadSegmentResponse ::= SEQUENCE {
    load-data          OctetString,
    more-follow        BOOLEAN
    -- true if there are other segments to be saved
}
```

5.2.5.5 Definition of the confirmed terminate upload sequence service protocol

– Request parameters

TerminateUploadSequenceRequest ::= Identification

--See the constraints of the Initiate Upload Sequence service on the significance of "Identification".

"Identification" corresponds either to the domain name or Index or to the ULSM Index, depending on the context.--

– Response parameters

TerminateUploadSequenceResponse ::= NULL

5.2.5.6 Definition of the confirmed get domain attributes service protocol

– Request parameters

GetDomainAttributesRequest ::= Identification

– Response parameters

```
GetDomainAttributesResponse ::= SEQUENCE {
    list-of-identification SEQUENCE OF IDENTIFICATION
    object-class           ObjectClass
    -- Permitted value "Domain"
    list-of-capabilities   SEQUENCE OF OCTETSTRING,
    -- the content can be empty (caution the encoding of SEQUENCE OF continues)--
    state                  DomainState,
    predefined             BOOLEAN, --true if predefined--
    sub-mms-sharable       BOOLEAN, --true if sharable--
    list-of-pi             ListOfObjectId
    upload-in-progress      UNSIGNED-8,
    -- number of upload in progress--
    domain-access-protection DomainAccessProtection
    -- if the object is not protected its content shall be empty--
    extension              OCTETSTRING}
```

```
DomainState ::= UNSIGNED-8
{
    non-existent/predefined (0),
    loading                 (1),
    ready                   (2),
    in-use                  (3),
    complete                (4),
    incomplete              (5)
}
```

5.2.6 Program Invocation ASE protocol

5.2.6.1 General

This section defines the protocol concerning the following services:

- 1) CreateProgramInvocation,
- 2) DeleteProgramInvocation,
- 3) Start,
- 4) Stop,
- 5) Resume,
- 6) Reset,
- 7) Kill,
- 8) Get program invocation attributes.

NOTE The first 7 services can be executed locally.

5.2.6.2 Definition of the confirmed create program invocation service protocol

– Request_parameters

CreateProgramInvocationRequest ::= SEQUENCE{
 pi-identification Identification,
 list-of-domains-identification ListOfObjectID,
 reusable BOOLEAN,
 --true for reusable--
 pi-access-protection PiAccessProtection
 extension OCTETSTRING}

NOTE The "pi-access-protection" parameter is assigned to the PI object created by the service. It characterizes the level of protection of the object with respect to CLIENTS.

PiAccessProtection ::= BITSTRING{
 --For a quality of service 1 the content of BITSTRING shall be empty
 For a quality of service 2 the content of BITSTRING shall correspond to the following definition--
 --password--
 password-bit 8 (0),
 password-bit 7 (1),
 password-bit 6 (2),
 password-bit 5 (3),
 password-bit 4 (4),
 password-bit 3 (5),
 password-bit 2 (6),
 password-bit 1 (7),
 --access_group--
 access-group 8 (8),
 access-group 7 (9),
 access-group 6 (10),
 access-group 5 (11),
 access-group 4 (12),
 access-group 3 (13),
 access-group 2 (14),
 access-group 1 (15),
 --access object protection--
 --value (16) is reserved--
 Dg (17),
 --If the bit value = 1 and the client belongs to one of the groups specified in the "access groups" attribute then
 he can delete the PI by the Kill, Delete-PI services--
 Hg (18),
 --If the value of the bit = 1 and the client belongs to one of the groups specified in the "access groups"
 attribute, then he can stop the execution of the PI by the Stop service --
 Sg (19),
 --If the value of the bit = 1 and the client belongs to one of the groups specified in the "access groups"
 attribute, then he can start execution of the PI by the Start, Resume and Reset services --
 --value (20) is reserved--
 D (21),
 --If the value of the bit = 1 and the client has provided a password identical to that of the "password" attribute
 he can delete the PI by the Kill, Delete-PI services--
 H (22),

```

--If the value of the bit = 1 and the client has provided a password identical to that of the "password" attribute,
then he can stop execution of the PI by the Stop service--
S                                     (23),
--If the value of the bit = 1 and the client has provided a password identical to that of the "access group"
attribute, he can start execution of the PI by the Start, Resume and Reset services--
--The values (24), (25), (26), (27), (28) are reserved--
Da                                     (29),
--If the value of the bit = 1 all the clients can delete the PI by the Kill, Delete-PI services--
Ha                                     (30),
--If the value of the bit = 1 all the clients can stop execution of the PI by the Stop service--
Sa                                     (31)
--If the value of the bit = 1 all the clients can start execution of the PI by the Start, Resume and Reset
services--
}
}

```

– Response parameters

CreateProgramInvocationResponse ::= Index

The server can inform the client that he accepts to assign the proposed identifier to the created object by sending back in its response an UNSIGNED_16 type index with a non-significant value (Value = 32K).

The server can signify to the client that it is assigning to the created object an identifier of the index type by sending back in its response an index with a significant value (Value between 0 and 32K-1); this identifier replaces the proposed identifier if it is of the index type, otherwise the created object supports the name and the index as identifier.

5.2.6.3 Definition of the confirmed delete program invocation service protocol

– Request parameters.

DeleteProgInvocRequest ::= Identification.

– Response parameters.

DeleteProgInvocResponse ::= NULL

5.2.6.4 Definition of the confirmed start service protocol

– Request parameters

```

StartRequest ::= SEQUENCE{
    pi-identification          Identification,
    execution-argument         OCTETSTRING }
--the content of the execution argument can be empty--

```

– Response parameters

StartResponse ::= NULL

5.2.6.5 Definition of the confirmed stop service protocol

– Request parameters

StopRequest ::= Identification

– Response parameters

StopResponse ::= NULL

5.2.6.6 Definition of the confirmed resume service protocol

– Request parameters

```
ResumeRequest ::= SEQUENCE {
    pi-identification          Identification,
    execution-argument         OCTETSTRING }
--the content of the execution argument can be empty--
```

- Response parameters

```
Resume Response ::= NULL
```

5.2.6.7 Definition of the confirmed reset service protocol

- Request parameters

```
ResetRequest ::= Identification
```

- Response parameters

```
ResetResponse ::= NULL
```

5.2.6.8 Definition of the confirmed kill service protocol

- Request parameters

```
KillRequest ::= Identification
```

- Response parameters

```
KillResponse ::= NULL
```

5.2.6.9 definition of the confirmed service protocol get program invocation attributes

- Request parameters

```
GetProgramInvocationAttributesRequest ::= Identification
```

- Response parameters

```
GetProgramInvocationAttributesResponse ::= SEQUENCE {
    list-of-identification      SEQUENCE OF Identification
    object-class                ObjectClass --Permitted value "PI"--
    pi-state                    PIState
    list-of-domain-id           ListOfObjetid
    sub-mms-deletable           BOOLEAN, --true if deletable
    reusable                    BOOLEAN, --true if reusable
    execution-argument          OCTETSTRING,
    --the content of OCTETSTRING can be empty, otherwise it should contain the last value supplied by the Start
    or Resume service executed with success--
    pi-access-protection        PIAccessProtection
    --the content should be empty if the object is not protected
    extension                   OCTETSTRING }

PISTATE ::= UNSIGNED-8 {
    non-existent                (0)
    unrunnable                  (1)
    idle                         (2)
    running                     (3)
    stopped                     (4)
    starting                     (5),
    stopping                     (6)
    resuming                     (7)
    resetting                     (8)
}
```

5.2.7 Variable ASE protocol

5.2.7.1 Summary

This section defines the following service parameters:

- 1) Read (a variable-list),
- 2) Write (a variable-list),
- 3) InformationReport (a variable-list),
- 4) DefineVariableList (a variable-list),
- 5) DeleteVariableList (a variable-list),
- 6) Get Variable Access Attributes (a variable),
- 7) Get Variable List Attributes (a variable-list).

5.2.7.2 Common definitions

The following production is used by the Read (without result), Write and Information Report services--

```
VariableDescription::= CHOICE {
    variable_list [0]          IMPLICIT Identification,
    list of variables          [1] IMPLICIT SEQUENCE OF VariableAccessDescription
}
```

--The following production is used only by the Read service (with result)--

```
VariableDescriptionWithType::= CHOICE {
    variable_list [2]          IMPLICIT Identification,
    list of variables          [3] IMPLICIT SEQUENCE OF VariableAccessDescription
}
```

--The following production is used by the Read, Write and Information Report services. It allows global or partial access to variables identified by their index or their name--

```
VariableAccessDescription::= CHOICE {
--global-access-choice
    object-index          UNSIGNED-16 (0..32K-1),
    object-name           CHOICE {
        undefined-length-name [0] IMPLICIT OCTETSTRING SIZE (0..32),
        CS-fixed-length-name  [5] IMPLICIT OCTETSTRING SIZE (X)-- see note 1
    }
--partial access choice
--list-of-comp/elem-selection [(n≥1) th level nesting]
    object-index/list-of-selection [1] IMPLICIT SEQUENCE {
    object-index          UNSIGNED-16 (0..32K-1),
    list-of-selection     [0] IMPLICIT Selections },
    object-name/list-of-selection CHOICE {
        undefined-length-name/list-of-selection [2] IMPLICIT SEQUENCE {
            undefined-length-name OCTETSTRING SIZE (0..32),
            list-of-selection     [0] IMPLICIT Selections },
        cs-fixed-length-name/list-of-selection [6] IMPLICIT SEQUENCE {
            cs-fixed-length-name OCTETSTRING SIZE (X)—see note 1
            list-of-selection     [0] IMPLICIT Selection },
        },
    --one-comp/elem-selection [1 st level nesting]
    object-index/one-selection [3] IMPLICIT SEQUENCE {
        object-index          UNSIGNED-16 (0..32K-1),
        one-selection         UNSIGNED-16 },
    object-name-one-selection CHOICE {
        undefined-length-name/one-selection [4] IMPLICIT SEQUENCE {
            undefined-length-name OCTETSTRING SIZE (0..32),
            one-selection         UNSIGNED-16 },
        cs-fixed-length-name/one-selection [7] IMPLICIT SEQUENCE {
            cs-fixed-length-name OCTETSTRING SIZE (X)—see note 1
            one-selection         UNSIGNED-16 },
        },
    }
}
```

NOTE 1 CS-fixed-length-name can only be used if the Companion Standards have defined a value at X ($X \leq 32$). In this case, the length of OCTETSTRING SIZE (X) is no longer encoded.

```

Selections ::= SEQUENCE {
    alternate access          SEQUENCE OF {--the index number shall be ≥ 0.--
    successive deeper comp/elem index  UNSIGNED-16
    }
    access                   CHOICE {
        list of comp/elem index  [0] IMPLICIT SEQUENCE OF UNSIGNED-16
        range of elements       [1] IMPLICIT SEQUENCE {
            start element index  UNSIGNED-16
            number of elements   UNSIGNED-16
        }
    }
}
    
```

NOTE 2 A table is composed of several elements. Each element can be of the simple, table or structure type. Each element is identified by an index of the UNSIGNED-16 type. The first element is identified by an index with the value 0, the second element having the value 1, etc. The composition of a table is assumed to be known by all the clients.

NOTE 3 A structure is composed of several fields. Each field can be of the simple, table or structure type. Each field is identified by an index of the UNSIGNED-16 type. No rule is imposed on the values to be assigned to the field identifiers. The composition of a structure and the values of the field identifiers are assumed to be known by all the clients.

5.2.7.3 Definition of the confirmed read service protocol

– Request parameters

```

ReadRequest ::= CHOICE {
    without result variable access specification  VariableDescription
        --This choice indicates a variable-list to be read. It also indicates that the response shall not contain
        descriptors of the types of values read--
    with result variable access specification      VariableDescriptionWithType
        --This choice indicates a variable-list to be read. It also indicates that the response shall contain the
        descriptors of the type of variables--
}
    
```

– Response parameters.

-- The read procedure for a variable executed with success results in returning the variable value. An unsuccessful read procedure for a variable results in returning the ErrorType value--

```

ReadResponse ::= SEQUENCE {
    list-of-access-result      ListOfAccessResult,
        --ListOfAccessResult is of the BITSTRING type. Bit (0) is assigned to the first variable read, bit (1) to
        the second variable read and so on. For each variable read, the bit assigned to it indicates if the
        read operation has been executed successfully or has been unsuccessful. A bit set to 0 indicates
        that the read attempt has been unsuccessful--
    list-of-Type-Data-Specif  ListOfObjectType ,
        --ListOfObjectType does not contain any descriptor if the request does not require their restitution.
        In other cases, it is composed of descriptors of the read variables. If the reading of a variable is
        executed with success, the descriptor should indicate the type of the value read. On the contrary if it
        ends by a failure, the descriptor should indicate the type of the ErrorType value.
        If all the readings of variables are successfully executed, and the values read are of the same type,
        ListOfObjectType can then contain a single descriptor describing the type in question. The
        authorized types are those specified by "TypeDataSpecification" (see section 5.2.3.1.11)--
    list-of-Data-value        ListOfObjectValue
        --The content of ListOfObjectValue is composed of the data of the variables read. These data are
        juxtaposed. If the reading of a variable is executed with success, the data should be of the value
        read. (see "Data Value" section 5.2.3.1.12 for the different possible values). If the reading of a
        variable ends in failure, the data shall be of the ErrorType value. (see section 5.2.3.1.10)--
}
    
```

5.2.7.4 Definition of the confirmed write service protocol

– Request parameters.

```
WriteRequest ::= SEQUENCE {
    variables access specification      VariableDescription,
    --VariableDescription indicates a variable class object identifier list or a variable-list class object
    identifier to be written--
    list-of-Type-Data-Specif. ListOfObjectType,
    --The content of ListOfObjectType could be without any type descriptor.
    If not, it should contain the descriptor of each of the values of the variables to be written.
    If it contains a single descriptor, this signifies that the values of the variables to be written are all of
    the type indicated by the descriptor. The authorized types are those specified by
    "TypeDataSpecification" (see section 5.2.3.1.11)--
    list-of-Data-Value                ListOfObjectValue}
    --The content of ListOfObjectValue shall correspond to the values of the variables to be written. These
    values are juxtaposed. The permitted values are those specified by "Data Value" (see section 5.2.3.1.12)-
```

– Response parameters.

```
WriteResponse ::= SEQUENCE {
    list-of-access-result              ListOfAccessResult,
    --ListOfAccessResult is of the BITSTRING type. The bit (0) is assigned to the first written variable, the
    bit (1) is assigned to the second written variable and so on. For each written variable, the bit which is
    assigned to it indicates if the write operation has been successfully executed or that it was unsuccessful.
    A bit set to value 0 indicates that the write operation was unsuccessful--
    list-of-ErrorType                  ListOfErrorType
    --ListOfErrorType contains no value if all the writing operations of the variables have been successfully
    executed. If not, it contains the juxtaposition of ErrorType values of all the writings of variables which
    ended unsuccessfully. The first value should correspond to the first failure, the second to the second
    failure and so on (for the ErrorType value See section 5.2.3.1.10)--
}
```

5.2.7.5 Definition of the unconfirmed information report service protocol

– Request parameters

```
InformationReportRequest ::= SEQUENCE {
    local detail                       OCTETSTRING,
    --the content of the OCTETSTRING can be empty--
    variable access specification      VariablesDescription,
    --VariableDescription indicates a list of variable class object identifiers or a variable list class object
    identifier whose values are provided in the ListOfObjectValue parameter--
    list-of-Type-Data-Specif. ListOfObjectType,
    --the content of ListOfObjectType could be empty.
    Otherwise, it is composed of the descriptors of values present in the ListOfObjectValue parameter. If it
    contains only one descriptor this signifies that all the values of the "ListOfObjectValue" parameter are of
    the same type. The authorized types are those specified by "TypeDataSpecification" with the exception
    of error type (see section 5.2.3.1.10)--
    list-of-Data-Value                ListOfObjectValue }
    --The content of ListOfObjectValue is composed of the data of variables identified by the
    "VariablesDescription" parameter. These data are juxtaposed. The authorized values are those specified
    by "Data Value" (see section 5.2.3.1.11)--
```

5.2.7.6 Definition of the confirmed define variable list service protocol

– Request parameters

```
DefineVariableListRequest ::= SEQUENCE {
    proposed-Var-list-Identification Identification,
    --the proposed identifier can be either a name or an Index.--
    list-of-var-Identification          SEQUENCE OF Identification
    --Identifiers of the variables (no variable-list) to be linked to the created variable-list--
    variable-list access protection      VariableListAccessProtection,
    extension                           OCTETSTRING
    --allows characterizing the type of user, extension, etc.--
}
```

Review:

- The "variable-list access protection" parameter is assigned to the created Variable-list object. It characterizes the level of protection of the object with respect to CLIENTS.

```
VarListAccessProtection ::= BITSTRING{
    --for a service quality 1, the content of BITSTRING should be empty.
    For a service quality 2, the content of BITSTRING should correspond to the following definition--
    --password--
    password-bit 8,
    password-bit 7,
    password-bit 6,
    password-bit 5,
    password-bit 4,
    password-bit 3,
    password-bit 2,
    password-bit 1,

    --access group--
    access-group 8,
    access-group 7,
    access-group 6,
    access-group 5,
    access-group 4,
    access-group 3,
    access-group 2,
    access-group 1,

    --access object protection--
    --value (16) is reserved--
    Dg
    --If the bit value = 1 and the client has the "access group" rights required, he can delete the Variable-list by the
    Delete Variable-list service--
    Wg
    --If the bit value = 1 and the client has the "access group" rights required, he can write in the variables linked to
    the variable -list by the Write service (variable-list)--
    Rg
    --If the bit value = 1 and the client has the required "access group" rights, he can read the variables linked to
    the variable-list by the Read service (Variable-list)--
    --value (20) is reserved--
    D
    --If the bit value = 1 and the client has the required "password" right, he can delete the variable-list by the
    Delete variable-list service--
    W
    --If the bit value = 1 and the client has the required "password" rights, he can write in the variables linked to the
    variable-list by the Write service (Variable-list)--
    R
    --If the bit value = 1 and the client has the required "access group" right he can read the variables linked to the
    variable-list by the Read (Variable-list) service--
    --values (24), (25), (26), (27), (28) are reserved--
    Da
    --If the bit value = 1 all the clients can delete the variable list object by the Delete variable-list service--
    Wa
    --If the bit value = 1, all the clients can write the variables linked to the variable-list by the Write (Variable-list)
    service--
    Ra
    --If the bit value = 1 all the clients can read the variables linked to the variable-list by the Read(Variable-list)
    service--
}
```

– Response parameters

DefineVariableListResponse ::= Index

The server can indicate to the client that it accepts to assign the proposed identifier to the created object by returning in its response an index with a non-significant value (Value = 32K).

The server can indicate to the client that it is assigning an identifier of the index type to the created object by returning in its response an index with a significant value (Value between 0 and 32K-1); this identifier replaces the proposed identifier if it is of the index type, otherwise the created object supports both the name and the index as identifier--

5.2.7.7 Definition of the confirmed delete variable list service protocol

- Request parameters.

DeleteVariableListRequest ::= Identification.

"Identification" should be the identifier of the created object variable-list. The type of identification borne can be either a Name, or an Index, or a Name and an Index. See the explanations given by the DEFINE VARIABLE-LIST service.--

- Response parameters.

DeleteVariableListResponse ::= NULL

5.2.7.8 Definition of the confirmed get variable access attributes service protocol

- Request parameters.

GetVariableAccessAttributesRequest ::= Identification.

- Response parameters.

GetVariableAccessAttributesResponse ::= SEQUENCE {
 list-of-identification SEQUENCE OF IDENTIFICATION
 object-class Object-Class
 --Permitted value "variable"--
 type data description TypeDataSpecification,
 variableaccessprotection VariableAccessProtection
 extension OCTETSTRING }

VariableAccessProtection ::= BITSTRING{
 --the content shall be empty if the object is not protected, otherwise, it should contain the information defined below--

--password--
 password-bit 8 (0),
 password-bit 7 (1),
 password-bit 6 (2),
 password-bit 5 (3),
 password-bit 4 (4),
 password-bit 3 (5),
 password-bit 2 (6),
 password-bit 1 (7),
 --access group--
 access-group 8 (8),
 access-group 7 (9),
 access-group 6 (10),
 access-group 5 (11),
 access-group 4 (12),
 access-group 3 (13),
 access-group 2 (14),
 access-group 1 (15),
 --access object protection--
 --values (16), (17) reserved--
 Wg (18),
 --If the bit value = 1 and the client belongs to one of the groups specified in the "group access" attribute, he can write the variable by the Write service (variable) --
 Rg (19),
 --If the bit value = 1 and the client belongs to one of the groups specified in the "access group" attribute, he can read the variable by the Read service (Variable)--
 -- values (20), (21) are reserved--
 W (22),
 --If the bit value = 1 and the client has the "password" rights required, he can write the variable by the Write service (variable) --
 R (23),
 --If the bit value = 1 and the client has the "group access" rights required, he can read the variable by the Read service (Variable) --
 --values (24), (25), (26), (27), (28), (29) are reserved --
 Wa (30),
 --If the bit value = 1 all the clients can write the variable by the Write service (variable) -

Ra (31)
 --If the bit value = 1 all the clients can read the variable by the Read service (Variable)--
 extension OCTETSTRING}

5.2.7.9 Definition of the confirmed get variable list attributes service protocol

– Request parameters.

GetVariableListAttributesRequest ::= Identification.

– Response parameters.

GetVariableListAttributesResponse ::= SEQUENCE {
 list-of-identification SEQUENCE OF IDENTIFICATION
 object-class ObjectClass
 sub-mms-deletable BOOLEAN, --true if deletable
 list-of-variable ListOfObjectID,
 variable-list-access-protection VariableListAccessProtection
 --the content should be empty if the object is not protected.--
 extension OCTETSTRING }

VariableListAccessProtection ::= BITSTRING{
 --the content shall be empty if the object is not protected, otherwise, it should contain the information defined below--
 --password--
 password-bit 8 (0),
 password-bit 7 (1),
 password-bit 6 (2),
 password-bit 5 (3),
 password-bit 4 (4),
 password-bit 3 (5),
 password-bit 2 (6),
 password-bit 1 (7),
 --access_group--
 access-group 8 (8),
 access-group 7 (9),
 access-group 6 (10),
 access-group 5 (11),
 access-group 4 (12),
 access-group 3 (13),
 access-group 2 (14),
 access-group 1 (15),
 --access object protection--
 --value (16) is reserved--
 Dg (17)
 --If the bit value = 1 and the client belongs to one of the groups specified in the "Access Groups" attribute, then he can delete the Variable-List object by the Delete Variable-List service--
 Wg (18),
 --If the bit value = 1 and the client belongs to one of the groups specified in the "Access Groups" attribute, then he can write the variable by the Write (Variable-List) service--
 Rg (19),
 --If the bit value = 1 and the client belongs to one of the groups specified in the "Access Groups" attribute, then he can read the variable by the Read (Variable-List) service--
 -- value (20) is reserved--
 D (21)
 --If the bit value = 1 and the client has given a password identical to that specified in the "Password" attribute, he can delete the Variable-List object by the Delete Variable-List service--
 W (22),
 --If the bit value = 1 and the client has given a password identical to that specified in the "Password" attribute, he can write the variable by the Write (Variable-List) service.
 R (23),
 --If the bit value = 1 and the client has given a password identical to that specified in the "Password" attribute, he can read the variable by the Read (Variable-List) service--
 --values (24), (25), (26), (27), (28) are reserved --
 Da (29)
 --If the value of the bit = 1 all the clients can delete the Variable-List object by the Delete Variable-List service--
 Wa (30),
 --If the bit value = 1 all the clients can write the variable by the Write service (Variable list) -
 Ra (31)
 --If the bit value = 1 all the clients can read the variable by the Read service (Variable list)--
 }

5.2.8 Event ASE protocol

5.2.8.1 General

This section defines the protocol for the following services:

- 1) Event Notification,
- 2) Acknowledge Event Notification,
- 3) Alter Event Condition Monitoring,
- 4) Get Alarm Summary,
- 5) Get Event Condition Attributes.

5.2.8.2 Definition of the unconfirmed event notification service protocol

– Request parameters.

EventNotification ::= SEQUENCE {
 common-information CommonInformation
 list-of-event-identification ListOfObjectId,
 list-of-event-message-description SEQUENCE OF TypeDataSpecification
 --NOTE
 --The "list-of-event-message-description" parameter gives the list of descriptors of the "EventMessage"
 attributes of the objects listed in the ListOfObjectId parameter. This list could be reduced to ZERO descriptor.
 Each TypeDataSpecification describes the EventMessage type.
 Each "Event Message" is a structure having one or more fields according to the type of the event object.
 According to the type of the object, the description of an "EventMessage" shall be that of a structure having:
 a) either only one field:
 - this field describes Event Data. The authorized types are those defined by TypeDataSpecification with an
 exception of error type.
 b) or two fields:
 - the first field describes Event Occurrence ID. The authorized type is OCTETSTRING SIZE(2).
 - the second field describes Event Data. The authorized types are those defined by TypeDataSpecification
 excepting error type.
 c) or three fields:
 - the first field describes Event State. The authorized type is BITSTRING-16.
 - the second field describes Event Occurrence ID. The authorized type is OCTETSTRING SIZE(2).
 - the third field describes Event Data. The authorized types are those defined by TypeDataSpecification
 excepting error type.
 If all the "EventMessage" attributes are of the same type, the list could be reduced to a single "Event
 Message" descriptor--
 list-of-event-message-value OCTETSTRING
 --This parameter contains the juxtaposition of EventMessage values.
 NOTE
 Depending on the type of object, the value of an "EventMessage" comprises:
 a) a single field:
 this field gives the Event Data value. The permitted values are those defined by Data Value (see section
 5.2.3.1.12).
 b) or two fields:
 - the first gives the value of Event Occurrence ID. The value authorized is OCTETSTRING-2,
 - The second gives the value of Event Data. The values authorized are those defined by Data Value (see
 section 5.2.3.1.12),
 c) or three fields:
 - the first gives the value of Event State. The value permitted is BITSTRING-16,
 - the second gives the value of Event Occurrence ID. The value authorized is OCTETSTRING-2,

- the third gives the value of Event Data. The values permitted are those defined by Data Value (see section 5.2.3.1.12).--

CommonInformation ::= OCTETSTRING

--The contents of OCTETSTRING is composed of the ordered juxtaposition of the following information:

1- the value of the first information is a BITSTRING-16 and represents "Event Type". This information mandatory, indicates the type of event objects of which one event is included in the notification.

2- the value of the second information is an OCTETSTRING-2 and represents "EventTransactionOccurrenceID".

3- the value of the third information, optional, is known to the user; it occupies the rest of the octets of the OCTETSTRING content.--

```
EventType ::= BITSTRING-16 {
    common event detection support (0), --supported if the value is 1.
    acknowledge action support (1), --supported if the value is 1
    enable action support (2), --supported if the value is 1
    C.S/User action support (3 to n) --to be defined by C.S/User
}
```

EventTransactionOccurrenceID ::= OCTETSTRING-2

--the semantic is a local issue; it shall be described in the PICS.--

5.2.8.3 Definition of the confirmed acknowledge event notification service protocol

– Request parameters

This service enables acknowledgement of several object events. The acknowledgement of an object is done by forcing the object attribute Ack State to the value: True..The "Ack State" attribute is a component of the "Event State" attribute.--

```
AcknowledgeEventNotificationRequest ::= SEQUENCE {
    list of event identification ListOfObjectID
    list of event occurrence ID type
    description SEQUENCE OF TypeDataSpecification
    --The number of TypeDataSpecification can be null when the types are implicitly known. The
    description of each TypeDataSpecification shall be that of OCTETSTRING-2. Since all the type
    descriptors are identical, it is possible to reduce the list to a single descriptor--
    list of event occurrence Id value OCTETSTRING
    --The OCTET STRING content is composed of the ordered juxtaposition of values "EventOccurrenceID".
    Each of these values is an OCTETSTRING-2--
}
```

– Response parameters

```
AcknowledgeEventNotificationResponse ::= SEQUENCE {
    list-of-access-result ListOfAccessResult,
    --ListOfAccessResult is of the BITSTRING type. Bit (0) is assigned to the first acknowledged object, bit
    (1) is assigned to the second acknowledged object and so on. For each object acknowledged, the bit
    which is assigned to it indicates if the acknowledgement operation has been successfully carried out or
    has ended in a failure. The bit set to 0 signifies that the acknowledgement is a failure.
    list-of-ErrorType ListOfErrorType
    --the content of the ListOfErrorType is empty if all the acknowledgements have been carried out
    successfully, otherwise it contains the juxtaposition of the ErrorType values of all the acknowledgements
    which have ended in a failure. The first value shall correspond to the first failure, the second value shall
    correspond to the second failure and so on. The authorized Error Type values are those defined by the
    section 5.2.3.1.10--
}
```

5.2.8.4 Definition of the confirmed alter event condition monitoring service protocol

– Request parameters

This service enables modifying the attributes of Enable State or Extension States of several event objects. These attributes are the components of the "Event State" attribute. The C.S or the user can define other actions.

The modification of an "Event State" attribute can entail a new notification of the events of the same object. The present document does not indicate how or when this new notification took place--

```

AlterEventConditionMonitoringRequest::= SEQUENCE {
    list of event state action ID      SEQUENCE OF EventStatePathAccess
    --This parameter indicates the list of Event State attributes to be modified. For each modification it
    --indicates the identification of the object followed by the row of the action. See below the description of
    --the EventState with the possible actions--
    list of event state description    SEQUENCE OF TypeDataSpecification
    --The number of TypeDataSpecification can be null when the types are implicitly known. The
    --description of each "Event State", made by TypeDataSpecification, shall be that of BITSTRING-16. When
    --all the type descriptors are identical it is possible to reduce the list to a single descriptor--
    list of event state value          OCTETSTRING
    --The content of OCTET STRING is composed of the ordered juxtaposition of the "EventState" values.
    --Each Event State value is represented by a BITSTRING-16 value with the following constraints: the bits
    --which do not belong to the expected action shall be forced to 0. See below the definition of the "Event
    --State" actions--
}

EventStatePathAccess::= CHOICE {
    event-index/action-path-description [3] IMPLICIT SEQUENCE {
        event-index      UNSIGNED-16 (0..32K-1),
        action-rank      ActionRank
    },
    event-name/action-path-description CHOICE {
        undefined-length-name [4] IMPLICIT SEQUENCE {
            event-name      OCTETSTRING SIZE (0..32),
            action-rank      ActionRank
        },
        cs-defined-length-name [7] IMPLICIT SEQUENCE {
            event-name      OCTETSTRING SIZE (X),
            action-rank      ActionRank
        }
    }
}

```

NOTE CS-defined-length-name can only be used if the Companion Standards have defined the value $X \leq 32$. In this case the length of OCTETSTRING SIZE (X), is no longer encoded. (See encoding in appendix 10.A).

```

ActionRank::= UNSIGNED-16 {
    rank 0      (0), --reserved value
    rank 1      (1), --validation/invalidation
    rank 2 to n (2) to (n), --user action/CS
}

EventState::= BITSTRING-16 {
    acknowledgement (0), --true if the object is acknowledged
    validation       (1), --true if the object is validated
    extension-state  (2) to (p), --the user or the CS can lengthen the list of actions.
}

```

To this end, we recommend on one hand to break down the bits from 2 to p into n ordered sets, each including 1 or several bits and, on the other hand, to assign to each set an action of rank 2 to n--

– Response parameters

```

AlterEventConditionMonitoringResponse::= SEQUENCE {
    list-of-access-result      ListOfAccessResult,
    --ListOfAccessResult is of the BITSTRING type. Bit (0) is assigned to the first altered object, Bit (1) is
    --assigned to the second altered object and so on. For each altered object, the bit which is assigned to it
    --indicates if the alteration operation has been successfully carried out or if it ended in a failure. A bit
    --positioned to 0 value signifies that the modification ended in a failure--

    list-of-ErrorType          ListOfErrorType
    --the content of ListOfErrorType is empty if all the modifications have been successfully executed,
    --otherwise it contains the juxtaposition of the ErrorType values of all the modifications which have ended
    --in a failure. The first value shall correspond to the first failure, the second value shall correspond to the
    --second failure and so on. The authorized values for Error Type are those defined by section 5.2.3.1.10-
}

```

5.2.8.5 Definition of the confirmed get alarm summary service protocol

– Request parameters.

```
GetAlarmSummaryRequest::= CHOICE {
  without result events access      [1] IMPLICIT ListOfObjectID ,
    --This choice indicates a list of event objects whose Event_Message attribute is to be read. It also
    indicates that the response shall not contain descriptors of the read value type--
  with result events access        [3] IMPLICIT ListOfObjectID
    --This choice indicates a list of object events whose Event_Message attribute is to be read. It also
    indicates that the response shall contain the descriptors of the type of values read--
}
```

– Response parameters.

-- The reading procedure for a successfully executed Event_Message attribute causes returning its value. The reading procedure of an unsuccessful Event_Message attribute causes returning of the ErrorType value--

```
GetAlarmSummaryResponse::= SEQUENCE {
  list-of-access-result      ListOfAccessResult,
    --ListOfAccessResult is of the BITSTRING type. Bit (0) is assigned to the first attribute read, bit (1) is
    assigned to the second attribute read and so on. For each attribute read, the bit which is assigned to it
    indicates if the reading operation has been carried out successfully or has ended in a failure. A bit
    positioned at 0 value signifies that the reading is a failure--
  list-of-event-message-description SEQUENCE OF TypeDataSpecification --see note 1--
  list-of-event-message-value      OCTETSTRING --see note 2--
}
```

NOTE 1 This parameter contains a list of descriptors. This list could be reduced to ZERO descriptor if the request does not demand the descriptor types. Each TypeDataSpecification describes either an EventMessage type if reading operation was successful, or an Error Type if reading operations were a failure. If all the reading operations were carried out successfully and the descriptors were of the same type, the list could be reduced to only one descriptor of Event Message.

NOTE 2 Depending on the type of the object the "EventMessage" descriptor shall be that of a structure having:

- a) either only one field:
 - this field describes the Event Data type. The authorized types are those defined by TypeDataSpecification except error type.
- b) or two fields:
 - the first describes the Event Occurrence ID. The authorized type is OCTETSTRING-2,
 - the second describes the Event Data Type. The authorized types are those defined by TypeDataSpecification except error type.
- c) or three fields:
 - the first describes the Event State type. The authorized type is BITSTRING-16,
 - the second describes the type of Event Occurrence ID. The authorized type is OCTETSTRING-2,
 - the third describes the Event Data type. The authorized types are those described by TypeDataSpecification except error type.--

This parameter contains the juxtaposition of Event Message or Error Type values.

Each reading operation carried out successfully restores the Event Message value. Each reading operation ended in a failure restores Error Type value.

NOTE 3 According to the type of the object, the value of an "EventMessage" comprises:

- a) either only one field:
 - this field gives the Event Data value. The authorized values are those defined by Data Value (see 8.1.10.2).
- b) or two fields:
 - the first field gives the Event Occurrence ID value. The authorized value is OCTETSTRING-2,
 - the second gives the Event Data value. The authorized values are those defined by Data Value (see 8.1.10.2),
- c) or three fields:
 - the first field gives the Event State value. The authorized value is BITSTRING-16.
 - the second field gives the Event Occurrence ID value. The authorized value is OCTETSTRING-2,
 - the third field gives the Event Data value. The authorized values are those defined by Data Value (see 8.1.10.2).--

5.2.8.6 Definition of the confirmed get event condition attribute service protocol

– Request parameters.

GetEventConditionAttributesRequest ::= Identification

– Response parameters.

GetEventConditionAttributesResponse ::= SEQUENCE {
 list-of-identification SEQUENCE OF IDENTIFICATION
 object-class ObjectClass --permitted value "Event"--
 event type EventType
 --This parameter indicates the type of object. For the definition of Event Type, see the Event Notification
 service (section 8.7.2)--
 event-message-type-description TypeDataSpecification
 --This parameter gives the descriptor of "Event Message" of an object. The response of the "Get Alarm
 Summary" service (section 8.7.5) explains in detail how to build a description of Event Message--
 Event-access-protection ::= EventAccessProtection
 --The content of this parameter is empty if the object is not protected--
}

EventAccessProtection ::= BITSTRING {
 --The content is empty if the object is not protected, otherwise it shall correspond to the following description.
 --password--
 password bit 8 (0),
 password bit 7 (1),
 password bit 6 (2),
 password bit 5 (3),
 password bit 4 (4),
 password bit 3 (5),
 password bit 2 (6),
 password bit 1 (7),
 --access groups--
 access group 8 (8),
 access group 7 (9),
 access group 6 (10),
 access group 5 (11),
 access group 4 (12),
 access group 3 (13),
 access group 2 (14),
 access group 1 (15),
 --bit (16) reserved
 Dg (17),
 --if the value is 1 and if the client has the group access rights required he can validate/invalidate and read the
 object by means of the Alter Event Condition Monitoring and Get Alarm Summary services--
 Wg (18),
 --if the value is 1 and if the client has the group access rights required he can acknowledge and read the object
 by means of Acknowledge Event Notification and Get Alarm Summary services--
 --bits (19), (20) reserved
 D (21),
 --if the value is 1 and if the client provides a password identical to that specified in the "password" attribute, he
 can validate/invalidate and read the object by means of the Alter Event Condition Monitoring and Get Alarm
 Summary--
 W (22),
 --if the value is 1 and the client has the password rights required he can acknowledge and read the object by
 means of Acknowledge Event Notification and Get Alarm Summary services--
 --bits (23), (24), (25), (26), (27), (28) reserved
 Da (29),
 --if the value is 1 all the clients can validate/invalidate and read the object by means of Alter Event Condition
 Monitoring and Get Alarm Summary services--
 Wa (30),
 --if the value is 1 all the clients can acknowledge and read the object by means of Acknowledge Event
 Notification and Get Alarm Summary services--
 --bit (31) reserved
 extension OCTETSTRING}

6 Structure of protocol machines

Interface to FAL services and protocol machines are specified in this clause.

NOTE The state machines and ARPMs defined in this section only define the valid events for each. It is a local matter to handle these invalid events.

The behavior of the FAL is described by three integrated protocol machines. Specific sets of these protocol machines are defined for different AREP types. The three protocol machines are: FAL Service Protocol Machine (FSPM), the Application Relationship Protocol Machine (ARPM), and the Data Link Layer Mapping Protocol Machine (DMPM). The relationship among these protocol machines as well as primitives exchanged among them are depicted in Figure 9.

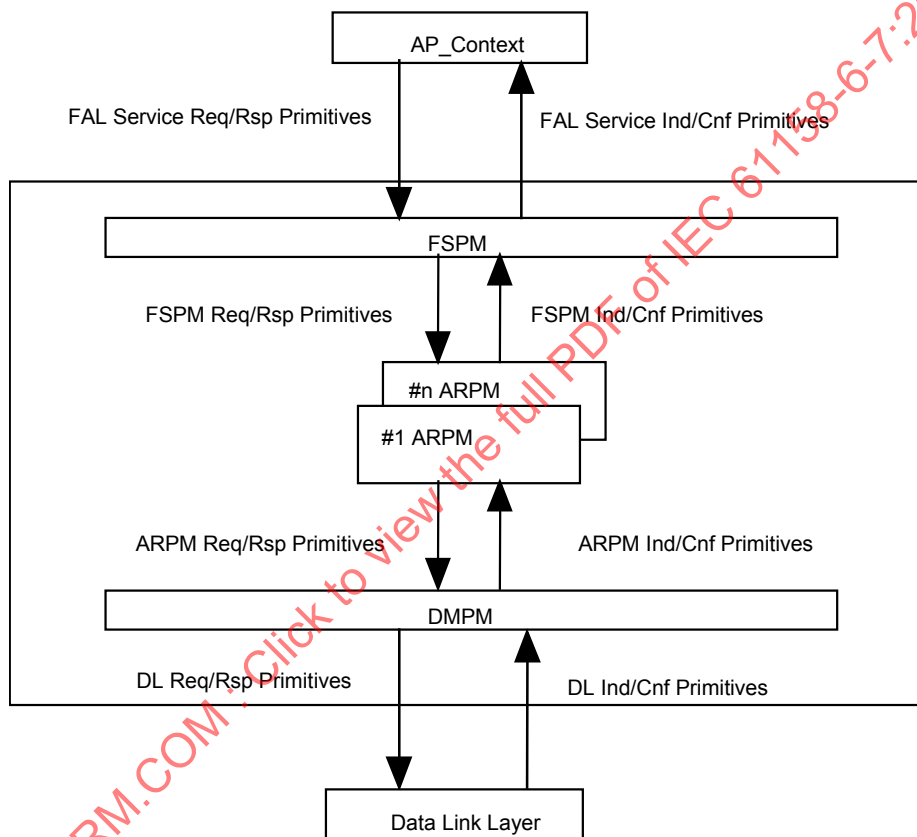


Figure 9 – Relationships among Protocol Machines and Adjacent Layers

The FSPM describes the service interface between the AP-Context and a particular AREP. The FSPM is common to all the AREP classes and does not have any state changes. The FSPM is responsible for the following activities:

- to accept service primitives from the FAL service user and convert them into FAL internal primitives;
- to select an appropriate ARPM state machine based on the AREP Identifier parameter supplied by the AP-Context and send FAL internal primitives to the selected ARPM;
- to accept FAL internal primitives from the ARPM and convert them into service primitives for the AP-Context;
- to deliver the FAL service primitives to the AP-Context based on the AREP Identifier parameter associated with the primitives.

The ARPM describes the establishment and release of an AR and exchange of FAL-PDUs with a remote ARPM(s). The ARPM is responsible for the following activities:

- a) to accept FAL internal primitives from the FSPM and create and send other FAL internal primitives to either the FSPM or the DMPM, based on the AREP and primitive types;
- b) to accept FAL internal primitives from the DMPM and send them to the FSPM as a form of FAL internal primitives;
- c) if the primitives are for the Establish or Abort service, it shall try to establish or release the specified AR.

The DMPM describes the mapping between the FAL and the DLL. It is common to all the AREP types and does not have any state changes. The DMPM is responsible for the following activities:

- a) to accept FAL internal primitives from the ARPM, prepare DLL service primitives, and send them to the DLL;
- b) to receive DLL indication or confirmation primitives from the DLL and send them to the ARPM in a form of FAL internal primitives.

For the Type-7 Application Layer protocol: the following restrictions apply.

- AP-Context State machine is not used.
- FAL Service protocol state machines are not used for MPS ASE but are used globally for Sub-MMS (VMD ASE, Domain ASE; Program Invocation ASE, Variable ASE, Event ASE, Directory ASE).
- Application Relationship Protocol Machine (ARPM) and Data Link Layer Mapping Protocol Machine (DMPM) are combined together for both MCS AR ASE and MPS ASE:
- One ARPM-DMPM protocol machine for MPS ASE,
- One ARPM-DMPM protocol machine for MCS AR ASE.

7 AP-context state machine

Feature not used

8 Sub-MMS FAL service protocol machine (FSPM)

8.1 General

For the needs of the fieldbus messaging, this section imposes the MCS lower layer defined in the standard.

8.2 Projection of the SUB-MMS PDUs on the MCS services

SUB-MMS pdu	MCS lower layer service
Confirmed Request PDU	(A-Data or A-Unidata) with/without ack.
Confirmed Response PDU	(A-Data or A-Unidata) with/without ack.
Confirmed Error PDU	(A-Data or A-Unidata) with/without ack.
Unconfirmed Request PDU	(A-Data or A-Unidata) with/without ack.
Reject PDU	(A-Data or A-Unidata) with/without ack.
Initiate Request PDU	A-Associate.Request,
Initiate Response PDU	A-Associate.Response(result +),
Initiate Error PDU	A-Associate.Response(result -),
Conclude Request PDU	A-Release.Request,
Conclude Response PDU	A-Release.Response(result +),
Conclude Error PDU	A-Release.Response(result -).

8.3 Projection of the SUB-MMS abort service on the MCS services

The SUB-MMS Abort service without PDU, shall therefore be directly projected on the MCS A-Abort service; consequently the following actions will be ensured by the supplier of the SUB-MMS service:

- a) on receiving an Abort request service (generated by the Sub-MMS user) it shall generate a A-Abort request towards the lower layer,
- b) on receiving an A-Abort service indication (generated by the lower layer) it shall generate an Abort indication towards the USER.

8.4 Construction of a SUB-MMS-PDU from a service primitive

On receiving a service primitive (request/response) other than "Abort" the SUB-MMS service supplier shall:

- a) construct a PDU in conformity with the SUB-MMS protocol,
- b) provide the built PDU to the lower layer via an MCS service in conformity with the Sub-MMS PDU projection requirements of the Sub-MMS PDU for the MCS services.

8.5 Extraction of a valid service primitive from a SUB-MMS-PDU

On receiving an MCS service primitive other than A_Abort the following actions are to be carried out in the order indicated by the SUB-MMS service supplier:

- a) check that the "user-data" parameter of the primitive contains a valid PDU (in conformity with the SUB-MMS protocol);
- b) if the received PDU is valid, extract a SUB-MMS service primitive and deliver it to the Sub-MMS user;
- c) if the received PDU is invalid and the transfer has been performed by means of an association being established (reception with success of "InitiateReqPDU" and no successful transmission of "InitiateRespPDU") then the two following actions can be carried out:
 - 1) generate an Abort indication and deliver it to the Sub-MMS,
 - 2) generate an "A-Abort" request to the MCS so that the latter frees the reserved resources. This request will not be transferred to the remote correspondent because the MCS is incapable of it (See MCS ASE);
- d) if the PDU is invalid and if the transfer of the PDU has been performed by means of a negotiated association, a predefined association or without association then IEC 61158-5-7, shall be complied with to execute the two following actions:
 - 1) construct a RejectPDU and deliver to the layer in compliance with the projection requirements of the Sub-MMS PDU for the MCS services,
 - 2) generate an indication of the Reject service and deliver it to the Sub-MMS user.

8.6 Negotiation of an abstract syntax and a transfer syntax commonly called presentation-context

This paragraph defines the negotiation in the following framework: the lower layer is MCS. It does not show the reductions made by the various intervenants during the negotiation. These reductions are explained in detail in a Sub-MMS Initiate service.

Review: the MCS standard authorises only one abstract syntax and only one transfer syntax per association.

Sequencing of an association opening

1 - Calling user:

The Sub-MMS user wishing to open an association with a remote partner generates a primitive Initiate.Request to the Sub-MMS service supplier.

2 - Sub-MMS service supplier of the calling user:

On receiving the primitive Initiate.Request initiated by the local Sub-MMS user, the Sub-MMS service supplier reserves the necessary resources, constructs an Initiate.Request PDU and generates towards the MCS layer the A_Associate.Request service with the User Data parameter = Initiate.Request PDU.

3 - Sub-MMS service supplier of the called user:

On receiving an A_Associate.Indication service initiated by the MCS layer, the Sub-MMS layer carries out the following actions as required:

- a) if it notes that the User Data parameter of A_Associate service does not contain a valid Initiate.RequestPDU or if it is incapable of accepting the opening of an association for various reasons, it generates a primitive A_Associate.Response to the MCS layer with the User Data = Empty and Result = negative parameters because of the user (the user is the Sub-MMS service supplier).
- b) if it notes that the User Data parameter of the A_Associate service contains a valid Initiate.Request PDU and accepts the opening of an association, then it reserves the necessary resources and generates the primitive Initiate.Indication to the Sub-MMS user.

4 - Called user:

The Sub-MMS user can either accept or refuse the association.

1) Acceptance of the association:

Following the reception of the primitive Initiate.Indication the Sub-MMS user generates towards the Sub-MMS service supplier a primitive Initiate.Response(+) and arranges for the future exchange of messages via this communication channel.

2) Refusal of the association:

Following the reception of the primitive Initiate.Indication the Sub-MMS user generates towards the Sub-MMS service supplier a primitive Initiate.Response(-) and it considers the proposed association inexistant.

5- Sub-MMS service supplier of the called user:

The behaviour of the layer varies according to the Sub-MMS user's response.

- 1) On receiving a primitive Initiate.Response(+), the Sub-MMS service supplier builds an Initiate.Response PDU and generates towards the MCS layer the A_Associate.Response service with the User Data = Initiate.ResponsePDU and Results = positive parameters and arranges for the future exchanges of messages via this communication channel.
- 2) On receiving a primitive Initiate.Response(-), the Sub-MMS service supplier frees the reserved resources, builds an Initiate.Error PDU and generates towards the MCS layer the A_Associate.Response service with the User Data = Initiate.ErrorPDU and Result = negative parameters because of the user and it considers the proposed association inexistent.

6- Sub-MMS service supplier of the calling user:

The behaviour of the Sub-MMS service supplier varies according to the response received.

- 1) On receiving an A_Associate.Confirm(+) primitive, the Sub-MMS service supplier carries out the following actions as required:
 - a) if it notes that the User Data parameter of the A_Associate service does not contain a valid Initiate.ResponsePDU, it ignores the service.

- b) if it notes that the A_Associate service user data parameter contains a valid Initiate.Response PDU, it generates the Initiate.Confirm(+) primitive towards the Sub-MMS user and makes arrangements for the future message exchanges via this communication channel.
- 2) On receiving an A_Associate.Confirm(-) primitive, the Sub-MMS service supplier carries out the following actions as required:
- a) if it notes that the A_Associate service user data parameter does not contain a valid Initiate.ErrorPDU, it ignores the service.
 - b) if it notes that the A_Associate service user data parameter contains a valid Initiate.ErrorPDU it generates the Initiate.Confirm(-) primitive towards the Sub-MMS user, frees the reserved resources and considers the proposed association inexistent.

7 - Called user:

The behaviour of the Sub-MMS user varies according to the response received.

- 1) Positive response:
Following the reception of the Initiate.Confirm(+) primitive the Sub-MMS user takes his dispositions for the future exchanges of messages via this communication channel.
- 2) Negative response:
Following reception of the Initiate.Confirm(-) primitive the Sub-MMS user considers the proposed association inexistent.

NOTE 1 If is necessary to check the state of a negotiated or predefined association in the absence of traffic the Sub-MMS service supplier can periodically generate an MCS_A_Data request service with the parameter User data = empty. The implementation of this mechanism of the client is a local issue.

NOTE 2 If is necessary to check the state of a negotiated or predefined association in the absence of traffic the Sub-MMS service supplier can demand the periodic reception of a Sub-MMS A_Data service request with the parameter User data = empty.

8.7 Identification of the SUB-MMS core abstract syntax

- 1) The abstract syntax of the CORE-SUB-MMS module is identified by an object-identifier and an object-descriptor. This identification characterises the abstract syntax of the Sub-MMS core.

The present document assigns the following values to them:

FIELD BUS ASN.1 object-identifier value:
{iso standard 9506 part-protocol (2)
SUB-MMS-abstract-syntax-version1 (4) }

FIELD BUS ASN.1 object-descriptor value:
"sub-mms-abstract-syntax-major-version1".

- 2) The SUB-MMS-General-module-1 is identified by an object identifier and an object-descriptor. This identification characterizes the type of ASE Sub-MMS because it enables generating different abstract syntaxes.

This document assigns to them the following values.

FIELD BUS ASN.1 object-identifier value:
{iso standard 9506 part_protocol (2)
sub-mms-general-module-version1 (5) }
FIELD BUS ASN.1 object-descriptor value:
"sub-mms-general-module-major-version1"

3-The transfer syntax relative to the abstract syntax of the CORE-SUB-MMS and the CS is identified by an object-identifier and an object-descriptor:

FIELD BUS ASN.1 object-identifier value:

{iso standard 8825 fieldbus-encoding (5) }.

FIELDBUS ASN.1 object-descriptor value:

"fieldbus-encoding-rules",

The major version of this document is 1.

The minor version of this document is 1.

This document permits other transfer syntaxes.

8.8 Identification of the application context name

For the specific use of the application layer, this document proposes the application layer containing "CORE-SUB-MMS" as ASE identified by an object-identifier and an object-descriptor.

This document assigns them the following values.

FIELDBUS ASN.1 object-identifier value:

{iso standard 9506 part-protocol (2)

sub-mms-application-context-version1 (6)}

FIELDBUS ASN.1 object-descriptor value:

"iso sub-mms" .

8.9 Identification of the ASE of the core abstract syntax and the transfer syntax

This subclause defines the following equivalences on behalf of MCS:

1- For the following object-identifier:

FIELDBUS ASN.1 object-identifier value (Sub-MMS general model):

{iso standard 9506 part_protocol (2)

sub-mms-general-module-version1 (5) }

we assign the binary value {0} for the "TYPE of ASE" parameter of the A_Associate service of MCS.

2 - For the following object-identifier (Sub-MMS core):

FIELDBUS ASN.1 object-identifier value:

{iso standard 9506 part-protocol (2)

SUB-MMS-abstract-syntax-version1 (4) }

we assign the binary value {0} for the "ABSTRACT SYNTAX" parameter of the A_Associate service of MCS.

3 - For the following object-identifier (Sub-MMS encoding rule):

FIELDBUS ASN.1 object-identifier value:

{iso standard 8825 fieldbus-encoding (5) }.

we assign the binary value {0} for the "TRANSFERT SYNTAX" parameter of the A_Associate service.

9 Association relationship protocol machine (ARPM)

These machines are defined in this part of standardization with the DLL Mapping Protocol Machine, see next chapter.

10 DLL mapping protocol machine (DMPM)

10.1 MPS ARPM and DMPM

10.1.1 General

The protocol procedures specify:

- the interactions between the MPS application services and link layer services;
- the interaction between the MPS application service elements and the link layer services. The service elements are characterized by application layer internal mechanisms that cannot be locally accessed by the primitives offered to the user.

The mapping between the parameters specifying link faults and the *Type of Error* parameters of the application layer service primitives is not specified. The error types described are simply recommendations.

10.1.2 Service procedures

10.1.2.1 General

The application services protocol procedures describe the interactions with the link services by use of evaluation nets.

Nevertheless the behaviour described by the evaluation nets does not presume the internal implementation of the MPS finite state machines services.

Furthermore, the service procedures describe the mapping on the link services in terms of primitives and parameters.

10.1.2.2 Local read service

10.1.2.2.1 Service description

The local read service only interacts with the link services when reading a non resynchronized variable. When a variable is resynchronized the service performs a read on the *Private Value* and *Private Status* attributes of the *Consumed Variable Local Image* application object, situated at the application layer level.

10.1.2.2.2 Evaluation net

The A_Readloc service evaluation net is shown in Figure 10.

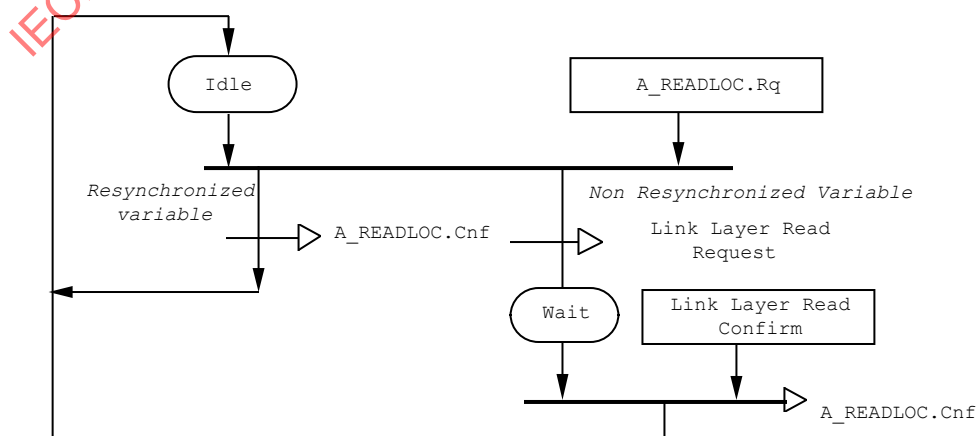


Figure 10 – A_Readloc service evaluation net

The confirmation associated with a A_READLOC.rq read request shall be negative whenever the data link confirmation was itself negative.

10.1.2.2.3 Mapping on link services

The mapping on the link layer services is as follows:

Link Layer Read Request: L_GET.request(IDENTIFIER)

Link Layer Read Confirmation: L_GET.confirm(IDENTIFIER,STATUS, L_DATA)

10.1.2.2.4 Mapping on link parameters

Mapping of MPS objects attributes on link services parameters is performed as follows:

IDENTIFIER: This parameter corresponds to the attribute *Identifier* of the *Consumed Variable Local Image* object.

L_DATA: This parameter corresponds to the transfer syntax PDU resulting from the encoding rules. The components of the PDU contents are available in the *Public value* and in the *Transmitted status value* attributes of the *Consumed Variable Local Image* object.

STATUS: When the confirmation is negative, this parameter specifies the error type.

10.1.2.3 Local write service

10.1.2.3.1 Service description

The local write service only interacts with the link services when writing a non resynchronized variable. When a variable is resynchronized the service writes the value of the *Private Value* attributes of the *Produced Variable Local Image* application object, situated at the application layer level.

10.1.2.3.2 Evaluation net

The A_WriteLoc service evaluation net is shown in Figure 11.

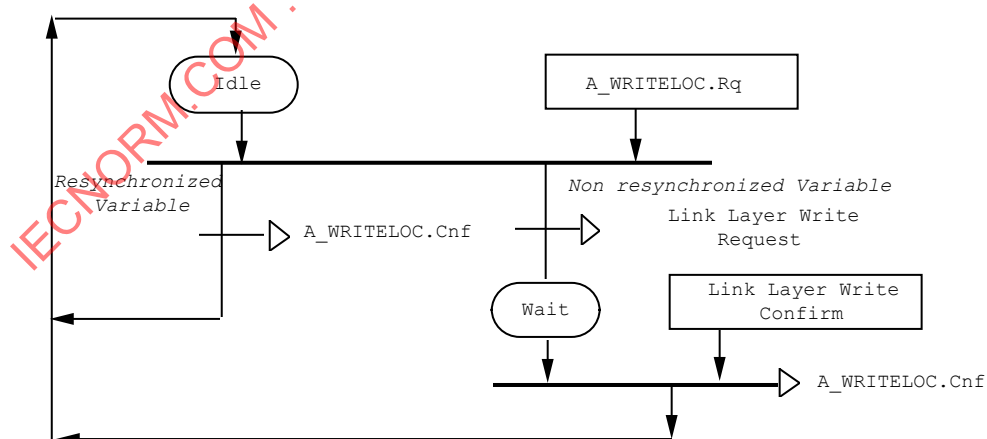


Figure 11 – A_WriteLoc service evaluation net

The confirmation associated with a A_WRITELOC.rq write request shall be negative whenever the data link confirmation was itself negative.

10.1.2.3.3 Mapping on the link services

The mapping on the link layer services is as follows:

Link Layer Write Request: L_PUT.request(IDENTIFIER, L_DATA)

Link Layer Write Confirmation: L_PUT.confirm(IDENTIFIER, STATUS)

10.1.2.3.4 Mapping on link parameters

Mapping of MPS objects attributes on link services parameters is performed as follows:

IDENTIFIER: This parameter corresponds to the attribute *Identifier* of the *Produced Variable Local Image* object.

L_DATA: This parameter corresponds to the transfer syntax PDU resulting from the encoding rules. The components of the PDU contents are available in the Public value and the Transmitted status value attributes of the Produced Variable Local Image object.

STATUS: When the confirmation is negative, this parameter specifies the error type.

10.1.2.4 Update service

10.1.2.4.1 Evaluation net

The A_Update service evaluation net is shown in Figure 12.

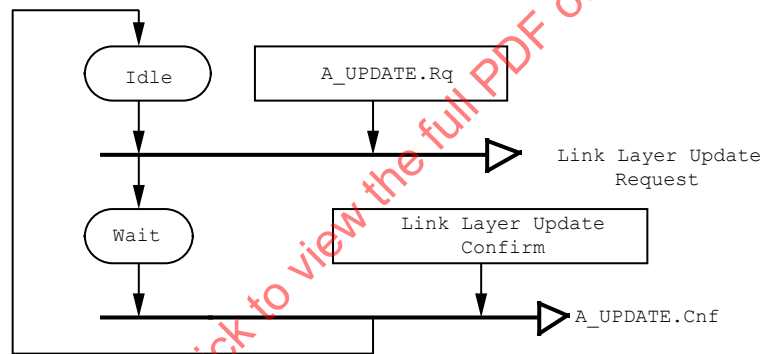


Figure 12 – A_Update service evaluation net

The confirmation associated with a A_UPDATE.rq update request shall be negative whenever the data link confirmation was itself negative.

10.1.2.4.2 Mapping on the link services

The mapping on the data link layer services is performed as follows:

Link Layer Update Request: L_FREE_UPDATE.request(SEQUENCE OF REQUESTED IDENTIFIERS, PRIORITY)

Link Layer Update Confirmation: L_FREE_UPDATE.confirm(STATUS)

10.1.2.4.3 Mapping on Link parameters

Mapping of MPS object attributes on link services parameters is performed as follows:

SEQUENCE OF REQUESTED IDENTIFIERS: This parameter corresponds to the attribute *Identifier* of one of the following objects: *Consumed Variable Local Image*, *Produced Variable Local Image* or *Third Part Variable Local Image* object. This attribute corresponds to the variable specified in the *Variable Specification* parameter of the A_UPDATE service primitive.

PRIORITY: This parameter corresponds to the *Priority* parameter of the A_UPDATE service primitive.

STATUS: When the confirmation is negative, this parameter specifies the error type.

10.1.2.5 Remote read service

10.1.2.5.1 Service description

When a variable is resynchronized, the remote read service performs only a link update request. The read value being the *Private value* attribute of the *Consumed Variable Local Image* object

When a variable is not resynchronized the remote read service first performs a link update request and, after confirmation of this request, the service performs a local read on the *Public value* attribute of the object *Consumed Variable Local Image*.

10.1.2.5.2 Evaluation net

The A_Readfar service evaluation net is shown in Figure 13.

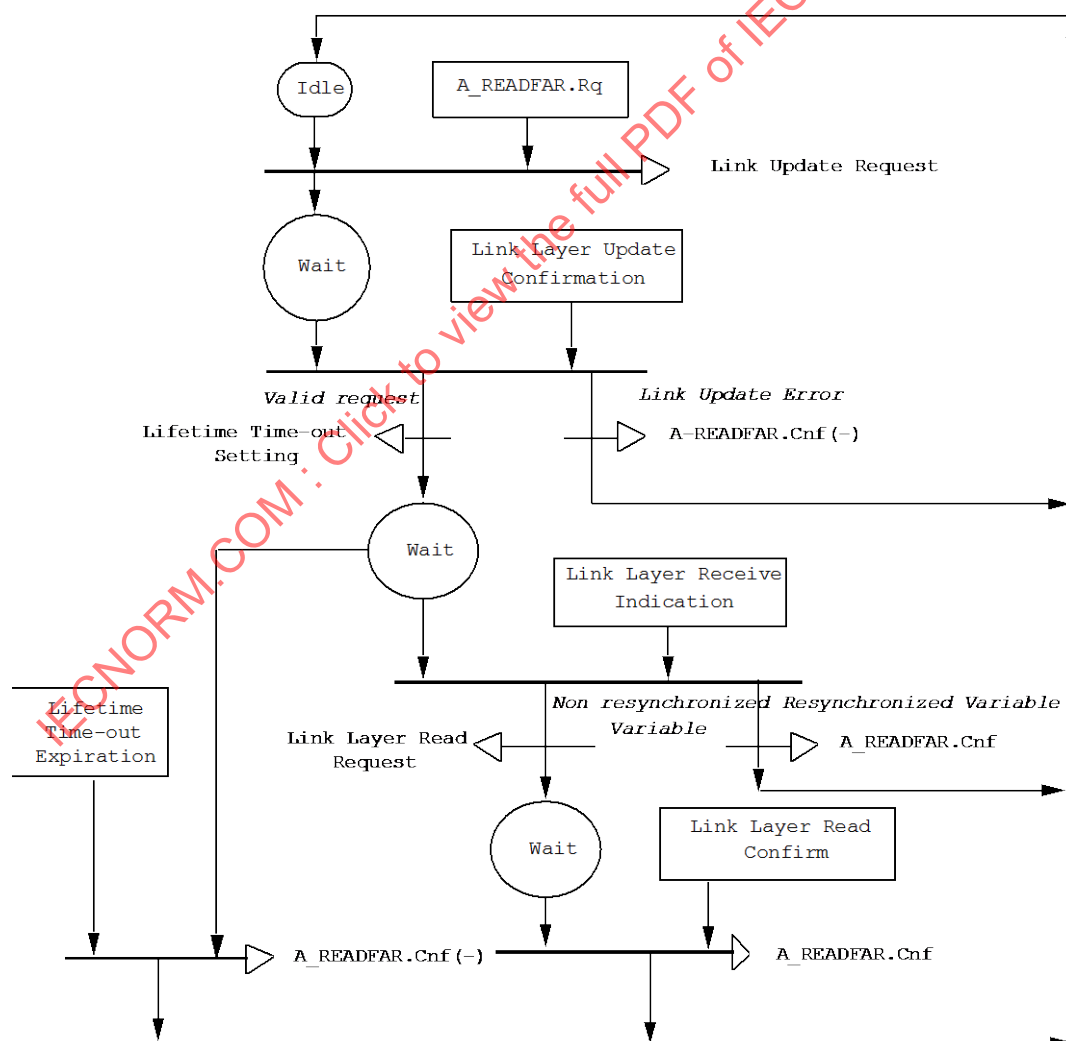


Figure 13 – A_Readfar service evaluation net

10.1.2.5.3 Mapping on the Link services

The mapping on the data Link Layer is made as follows:

Link Layer Read Request:	L_GET.(IDENTIFIER)
Link Layer Read Confirmation:	L_GET.confirm(IDENTIFIER,STATUS, L_DATA)
Link Layer Update Request:	L_FREE_UPDATE.request(SEQUENCE OF REQUESTED IDENTIFIERS, PRIORITY)
Link Layer Update Confirmation:	L_FREE_UPDATE.confirm(STATUS)
Link Layer Receive Indication:	L_RECEIVED.indication(IDENTIFIER)

10.1.2.5.4 Mapping on Link parameters

Mapping of MPS object attributes on link service parameters is performed as follows:

IDENTIFIER: This parameter corresponds to the attribute *Identifier* of the *Consumed Variable Local Image* object.

L_DATA: This parameter corresponds to the transfer syntax PDU resulting from the encoding rules. The components of the PDU contents are available in the *Public value* and in the *Transmitted status value* attributes of the *Consumed Variable Local Image* object.

SEQUENCE OF REQUESTED IDENTIFIERS: This parameter corresponds to the attribute *Identifier* of the *Consumed Variable Local Image* object.

PRIORITY: This parameter corresponds to the *Priority* parameter of the A_READFAR service primitive.

STATUS: When the confirm primitive is negative, this parameter specifies the type of the error.

10.1.2.6 Remote write service

10.1.2.6.1 Service Description

When a variable is resynchronized, the remote write service performs only a link update request, the value of the variable having been already write on the *Private value* attribute of the *Produced Variable Local Image* object

When a variable is not resynchronized the remote write service first performs a local write on the *Public value* attribute of the object *Produced Variable Local Image*, then after confirmation of the local write, the service performs a link update request.

10.1.2.6.2 Evaluation net

The A_writefar service evaluation net is shown in Figure 14.

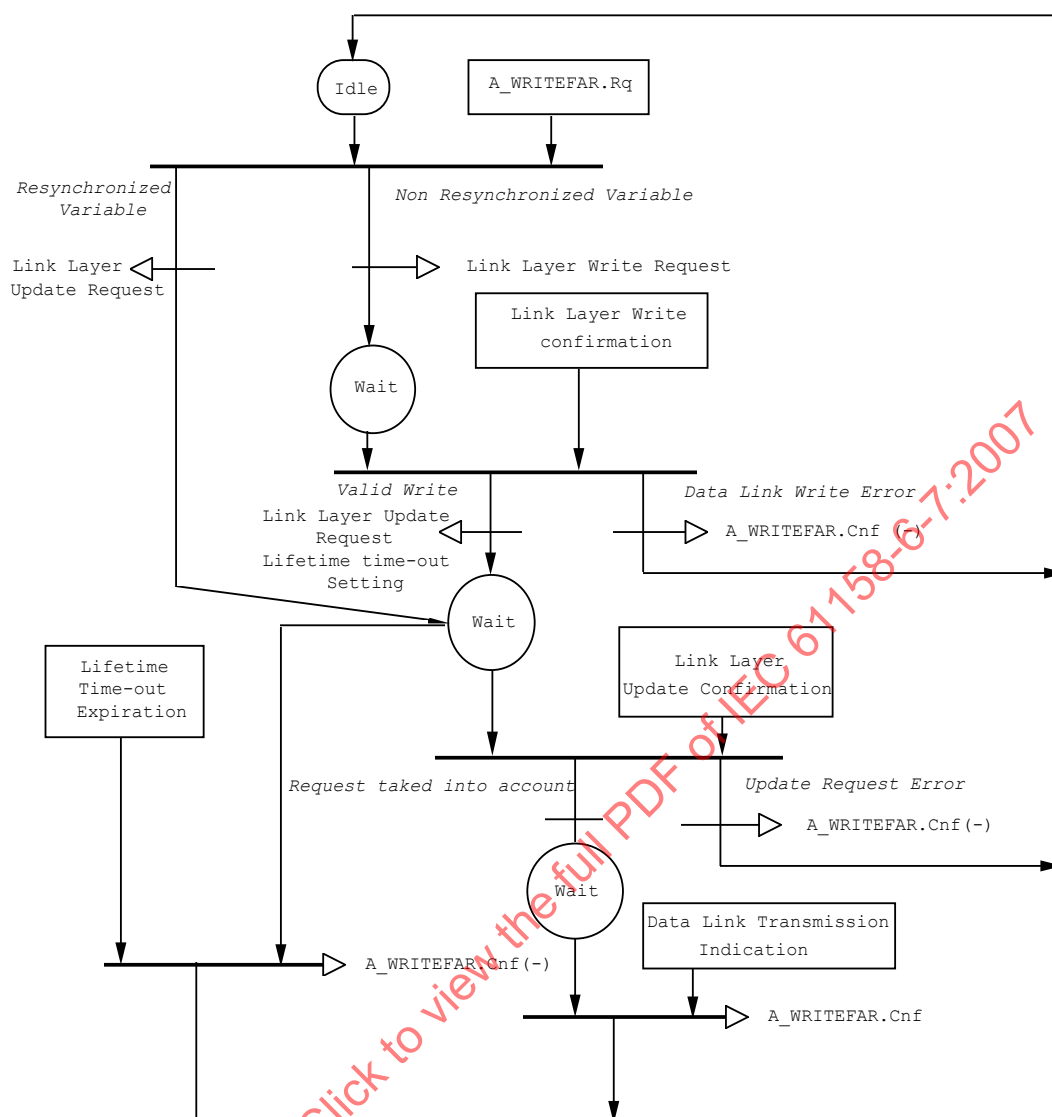


Figure 14 – A_Writefar service evaluation net

10.1.2.6.3 Mapping on the Link services

The mapping on the data Link Layer is performed as follows:

Link Layer Write Request:	L_PUT.request(IDENTIFIER, L_DATA)
Link Layer Write Confirmation:	L_PUT.confirm(IDENTIFIER, STATUS)
Link Layer Update Request:	REQUESTED IDENTIFIERS, PRIORITY)
Link Layer Update Confirmation:	L_FREE_UPDATE.confirm(STATUS)
Link Layer Transmission Indication:	L_SENT.indication(IDENTIFIER)

10.1.2.6.4 Mapping on the Link parameters

The mapping of MPS object attributes on link services parameters is performed as follows:

IDENTIFIER: This parameter corresponds to the attribute *Identifier* of the *Produced Variable Local Image* object

L_DATA: This parameter corresponds to the transfer syntax PDU resulting from the encoding rules. The components of the PDU contents are available in the *Public value* and in the *Transmitted status value* attributes of the *Produced Variable Local Image* object.

SEQUENCE OF REQUESTED IDENTIFIERS: This parameter corresponds to the attribute *Identifier* of the *Produced Variable Local Image* object.

PRIORITY: This parameter corresponds to the *Priority* parameter of the A_WRITEFAR service primitive.

STATUS: When the confirm primitive is negative, this parameter specifies the type of error.

10.1.2.7 Transmission Indication Service

10.1.2.7.1 Evaluation net

The A_Sent service evaluation net is shown in Figure 15.

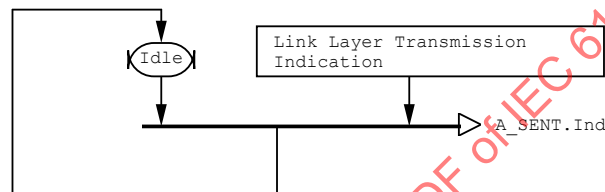


Figure 15 – A_Sent service evaluation net

10.1.2.7.2 Mapping on Link services

The mapping on the Link Layer services is performed as follows:

Link Layer Transmission Indication: L_SENT.indication(IDENTIFIER)

10.1.2.7.3 Mapping on the Link parameters

The mapping of MPS object attributes on link services parameters is performed as follows:

IDENTIFIER: This parameter corresponds to the attribute *Identifier* of the *Produced Variable Local Image* object of the variable which implements the transmission indication service.

10.1.2.8 Reception indication service.

10.1.2.8.1 Evaluation net

The A_Received service evaluation net is shown in Figure 16.

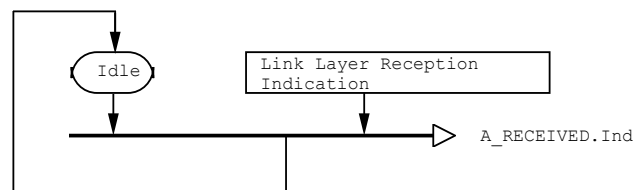


Figure 16 – A_Received service evaluation net

10.1.2.8.2 Mapping on the Link services.

The mapping on the Link Layer services is performed as follows:

Link Layer Reception Indication: L_RECEIVED.indication(IDENTIFIER)

10.1.2.8.3 Mapping on the Link parameters

Mapping of MPS objects attributes on link services parameters is performed as follows:

IDENTIFIER: This parameter corresponds to the attribute *Identifier* of the *Consumed Variable Local Image Object* of the variable which implements the reception indication service.

10.1.3 Service elements procedures

10.1.3.1.1 General

The application services elements are defined by Application Layer internal mechanisms that cannot be accessed by dedicated application primitives.

The protocol procedures associated with service elements describe the interaction with the link services by using evaluation nets; besides, the service procedures describe the mapping of those mechanisms on the link services of the AL TYPE 7 environment in terms of primitives and parameters.

10.1.3.2 Mapping of request and actions of the production validity status elaboration mechanisms:

An object *Produced Variable Local Image* has a service element procedure associated with the production validity status either the variable is resynchronized on production, or not resynchronized.

Status updating:

This action is mapped on the service primitive of data link level

L_PUT.request(IDENTIFIER,L_DATA)

The parameter IDENTIFIER, corresponds to the value of the attribute *Identifier* of the *Produced Variable Local Image* object.

The parameter L_DATA corresponds to the values of the attributes *Public Value* and *Transmitted Status Value* of the *Produced Variable Local Image* object.

Synchronization Order:

This request is mapped on a service primitive of data link level

L_RECEIVED.indication(IDENTIFIER) or

L_SENT.indication(IDENTIFIER)

The parameter IDENTIFIER, corresponds to the value of the attribute *Identifier* of the object *Consumed Variable Local Image* (of synchronization) or *Produced Variable Local Image* object (of synchronization) associated with the synchronization variable involved in the production validity status elaboration.

When the two objects *Consumed Variable Local Image* and *Produced Variable Local Image* associated with the synchronization variable are resident in the same application entity, the synchronization order is mapped on the consumed variable reception.

10.1.3.3 Mapping of request, and actions of the transmission validity status elaboration mechanisms:

A *Consumed Variable Local Image* object has a service element procedure associated with the transmission validity status whether the variable is resynchronized or not.

Variable reception:

This request is mapped on the data link level service primitive

L_RECEIVED.indication(IDENTIFIER)

The parameter IDENTIFIER corresponds to the value of the attribute *Identifier* of the *Consumed Variable Local Image* object.

Synchronization order:

This request is mapped on the link layer level services primitives

L_RECEIVED.indication (IDENTIFIER) or

L_SENT.indication(IDENTIFIER)

The parameter IDENTIFIER, corresponds to the value of the attribute *Identifier* of the object *Consumed Variable Local Image* (of synchronization) or *Produced Variable Local Image* object (of synchronization) associated with the synchronization variable involved in the transmission validity status elaboration mechanism.

10.1.3.4 Mapping of request, and actions of the recovery list elaboration mechanisms

The contents of the recovery list is managed by one service element which takes into account firstly, the synchronization order associated with the list, and secondly, the reception of the various variables of the list.

Recovery List Content Management

List Content Initialization:

These actions are mapped on the Data Link level service primitive

L_SPEC_UPDATE.request(SPECIFIED IDENTIFIER, SEQUENCE OF REQUESTED IDENTIFIERS)

The SPECIFIED IDENTIFIER parameter corresponds to the attribute *Recovery List Identifier* of the *List of Variables Local Image* object.

The SEQUENCE OF REQUESTED IDENTIFIERS corresponds to the attribute *Recovery List Content* of the *List of Variables Local Image* object.

List Synchronization Order:

This action is mapped on the data link level service primitive

L_RECEIVED.indication (IDENTIFIER)

The parameter IDENTIFIER corresponds to the attribute *Reference (synchronization) Variable* of the *Variables List Local Image* object.

When the two objects *Consumed Variable Local Image* and *Produced Variable Local Image* are resident in the same application entity, the synchronization order is mapped on the consumed variable reception.

Reception of the Variable List:

This action is mapped on the data link layer service primitive L_RECEIVED.indication (IDENTIFIER)

The parameter IDENTIFIER corresponds to the attribute *Reference Consumed Variable* of the *List of Variables Local Image* object.

NOTE When the attribute *Recovery Nature* of the *List of Variables Local Image* object has the value FRAGMENTED, the RQ_INHIBED link layer indicator shall have the value FALSE. When the attribute has the value INDIVISIBLE the indicator shall have the value TRUE.

10.1.3.5 Mapping of request, and actions of the resynchronization mechanisms.

When a variable is resynchronized, the involved mechanism uses:

- resynchronization order.
- The transfer of the public value into the private value for a consumed resynchronized variable.
- The transfer of the private value into the public value for a produced resynchronized variable.

Resynchronization Order:

This request is mapped on the data link layer level service primitives

L_RECEIVED.indication(IDENTIFIER) or
L_SENT.indication(IDENTIFIER)

The parameter IDENTIFIER, corresponds to the value of the attribute *Identifier* of the object *Consumed Variable Local Image* (of synchronization) or *Produced Variable Local Image* object (of synchronization), associated with the synchronization variable involved in the resynchronization mechanism.

When the two objects *Consumed Variable Local Image* and *Produced Variable Local Image* are resident in the same application entity, the synchronization order is the consumed variable reception.

Private Value into Public Value Transfer:

This action is mapped on the data link level service primitive

L_PUT.request(IDENTIFIER,L_DATA)

immediately after the update of the public context by means of the private context of the resynchronized variable.

The IDENTIFIER parameter corresponds to the attribute *Identifier* of the *Produced Variable Local Image* object.

The L_DATA parameter corresponds to the value of the attributes *Public Value* and *Transmitted Status Value* of the *Produced Variable Local Image* object.

Public Value into Private Value Transfer:

This action is mapped on the data link level service primitive

L_GET.request(IDENTIFIER,L_DATA)

immediately after the update of the private context by means of the public context of the resynchronized variable.

The IDENTIFIER parameter corresponds to the attribute *Identifier* of the *Consumed Variable Local Image* object.

The L_DATA parameter corresponds to the value of the attributes *Public Value* and *Transmitted Status Value* of the *Consumed Variable Local Image* object

10.1.3.6 Mapping of request, and actions of the consistency variable elaboration mechanisms

Each consistency variable value is elaborated in several application entities by using application service elements.

The value elaboration is done from the two following events: reception of the components of the variable list and reception of the synchronization variable of the list.

Consistency Variable Management

Consistency Variable Value Initialization:

These actions are mapped on the data link level service primitives L_SPEC_UPDATE.request (SPECIFIED IDENTIFIER, SEQUENCE OF REQUESTED IDENTIFIERS)

The parameter SPECIFIED IDENTIFIER corresponds to the attribute *Reference Produced Variable (of consistency)* of the object *List of Variable Local Image*.

The parameter SEQUENCE OF REQUESTED IDENTIFIERS corresponds to the attribute *Public Value* of the object *Produced Variable Local Image* which is referenced in the attribute *Reference Produced Variable (of consistency)* of the object *List of variables Local Image*.

Synchronisation List Order:

This action is mapped on the data link service primitive L_RECEIVED.indication(IDENTIFIER) or

L_SENT.indication(IDENTIFIER).

The parameter IDENTIFIER corresponds to the attribute *Reference (synchronization) Variable* of the object *List of variables Local Image*.

When the two objects *Consumed Variable Local Image* and *Produced Variable Local Image* are resident in the same application entity, the synchronization order is mapped on the consumed variable reception.

Reception of the Variable List:

This action is mapped on the data link level service primitive

L_RECEIVED.indication(IDENTIFIER)

The parameter IDENTIFIER corresponds to the attribute *Reference Consumed Variable* of the *List of Variables Local Image* object.

10.2 MCS ARPM and DMPM

10.2.1 General

The MCS protocols define the procedures which define the interactions between remote application entities and between data link layer entities, for the various services.

List of procedures:

- Association establishment
- Association termination
- Association revocation
- Associated mode data transfer
- Numbering
- Retry/Antiduplication
- Multiple anticipation
- Segmentation/Reassembly
- Non-associated mode data transfer
- Invocation identification

This definition, also features the different types of application data unit types implemented for carrying out the services.

- AARQ: Association establishment request
- AARP: Association establishment response
- RERQ: Association termination request
- RERP: Association termination response
- ABRQ: Association revocation request
- ATRQ: Associated transfer request
- AKAT: Associated transfer acknowledgement
- NTRQ: Non-associated transfer request
- AKNT: Non-associated transfer acknowledgement

NOTE Details of the coding of these application data units is covered in 3.8.6.

It is also important to know that all the protocol procedures consider the L_MSG_XX.rq/cnf/ind generic link primitives, and that it is only in 3.8.7 that an exact projection onto the link layer is to be found.

10.2.2 Association establishment

10.2.2.1 General

The association establishment procedure consists of two parts: the association establishment requesting element and responding element. These two elements of the procedure call upon functions which process the values of parameters and manage the resources of the association.

10.2.2.2 Requester element

a) For the association establishment requester element, the events and conditions accounted for are the following:

- invocation of the A_ASSOCIATE.rq() primitive by the user,
- reception of the AARP-APDU in the data of the L_MSG_XX.ind() primitive. This PDU contains the “Result” parameter which indicates whether establishment is accepted,
- a time-out which monitors the “Establishment duration” designed to limit the requester wait,
- invocation by the data link layer of the L_MSG_XX.cnf(+/-) primitive, indicating whether sending has taken place correctly or not,
- an establishment reinitialisation occurrence which corresponds to switching of the association status from OPEN to NON-EXISTENT.

NOTE This transition can be obtained by invoking the A_ABORT service by the user or the provider element, or any local initiative under network management responsibility.

- the association status value NON-EXISTENT governs acceptance of the A_ASSOCIATE.rq() primitive by the requester element.

b) The actions performed by this requester element are the following:

- invocation of the A_ASSOCIATE.cnf(+/-) primitive,
- transmission of an AARQ-APDU in the L_MSG_XX.rq() primitive data,
- resetting of the timer which monitors the “Establishment duration”, designed to limit the requester’s wait,
- assigns the value OPENING, ESTABLISHED or NON-EXISTENT to the association status.

c) Requester element mechanism:

An instantiation of this mechanism at the level of an application entity which supports the association establishment procedure causes the creation of an association. One or more associations can be in the process of being established simultaneously depending on the capability of the application entity.

The calling entity has invoked this mechanism for a given association.

In this mechanism, all interactions with the user are in the context of an “AE address” pair which identifies the two application entities concerned by the association opening, and all interactions with the data link layer are in the context of the link address (of ADAE type) which is associated with the AE address of the caller to localise the requester mechanism. See Figure 17.

NOTE It is important to understand that not all A_ASSOCIATE.rq() service invocation requests are necessarily followed by an A_ASSOCIATE.cnf(+/-) confirmation. Indeed, in the case of establishment reinitialisation, it is the A_ABORT.ind() primitive which follows the establishment request and thus takes the place of confirmation.

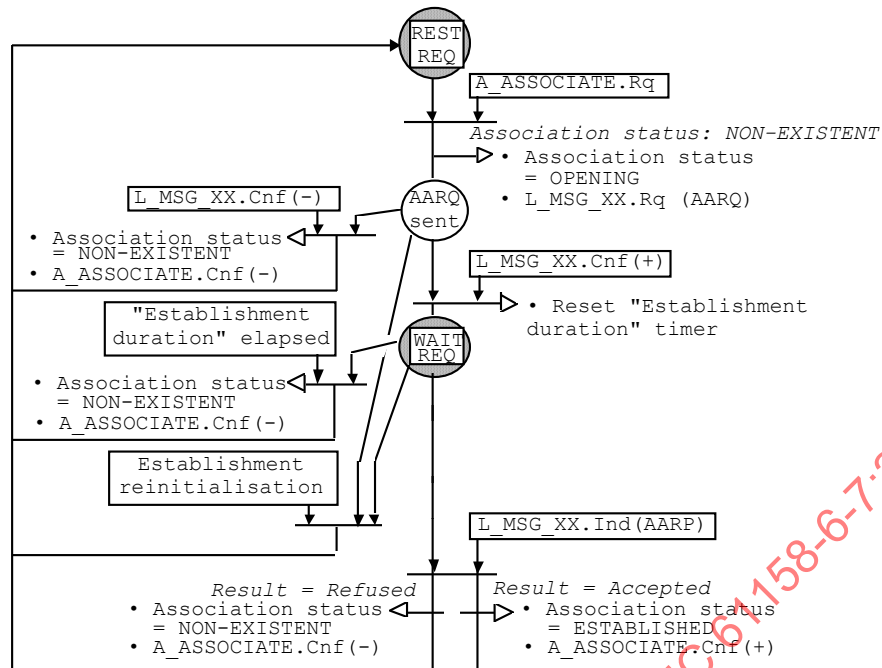


Figure 17 – Association establishment: Requester element state machine

d) Description of this mechanism

As soon as a A_ASSOCIATE.rq establishment request is made at a calling AEI access point for which there is no association (which corresponds to NON-EXISTENT association status), the latter switches to OPENING status and an establishment opening PDU is sent.

Sending consists of invoking the L_MSG_XX.rq() link service which is locally confirmed by L_MSG_XX.cnf() when the message has effectively disappeared from the link queues, subsequent to a successful transmission, or subsequent to a failure.

In the event of a link transmission failure, the requester element is reset to REST REQ status and the association to NON-EXISTENT status.

After sending an establishment request, the mechanism resets the "Establishment duration" timer and awaits an establishment response.

On arrival of the establishment response, the association changes to ESTABLISHED status or NON-EXISTENT status, depending on whether the establishment request has been accepted or refused by the responder to the request.

In addition, awaiting the response will be interrupted and the association set to NON-EXISTENT status if an establishment time-out is reached.

Finally, from the moment when the establishment procedure is started, i.e. when the requester element has "AARQ send" or "WAIT REQ" status, the latter is reinitialised in "REST REQ" status, when the status of the association changes from OPENING to NON-EXISTENT.

NOTE This change in the association status results from invoking the ABORT service by the local or remote user, or the local or remote provider element.

10.2.2.3 Responder element

a) For the establishment responder element, the events and conditions accounted for are the following:

- invoking the A_ASSOCIATE.rp() primitive by the user,
- receiving an AARQ-APDU in the L_MSG_XX.ind() primitive data,
- invoking the L_MSG_XX.cnf(+/-) primitive by the data link layer, indicating whether or not sending has taken place correctly,

- the occurrence of reinitialisation, which corresponds to changing the status of the association from OPENING to NON-EXISTENT,
- the association status value NON-EXISTENT governs acceptance of the receipt of an AARPQ by the responder element.

b) The actions carried out by this responder element are the following:

- transmission of an AARP-APDU in the L_MSG_XX.rq() primitive data,
- assigning values OPENING, ESTABLISHED or NON-EXISTENT to the association status.

c) Responder element mechanism:

An instantiation of this mechanism at the level of an application entity which supports the association establishment procedure causes the creation of an association. One or more associations can be in the process of being established simultaneously depending on the capacity of the application entity.

NOTE Consequently, all establishment requests arriving after the maximum number of simultaneous establishment requests is reached are lost and there is no corresponding response. When such a loss occurs, the requesters wait until establishment time-out before repeating their requests.

The called entity is that which invokes the mechanism for a given association.

In this mechanism, all interactions with the user take place in the context of a pair of “AE addresses” which identify the two application identities concerned by the association opening, and all interactions with the data link layer are in the context of a link address (of ADAE type) which is associated with the AE address of the called entity for localising the responder mechanism. See Figure 18.

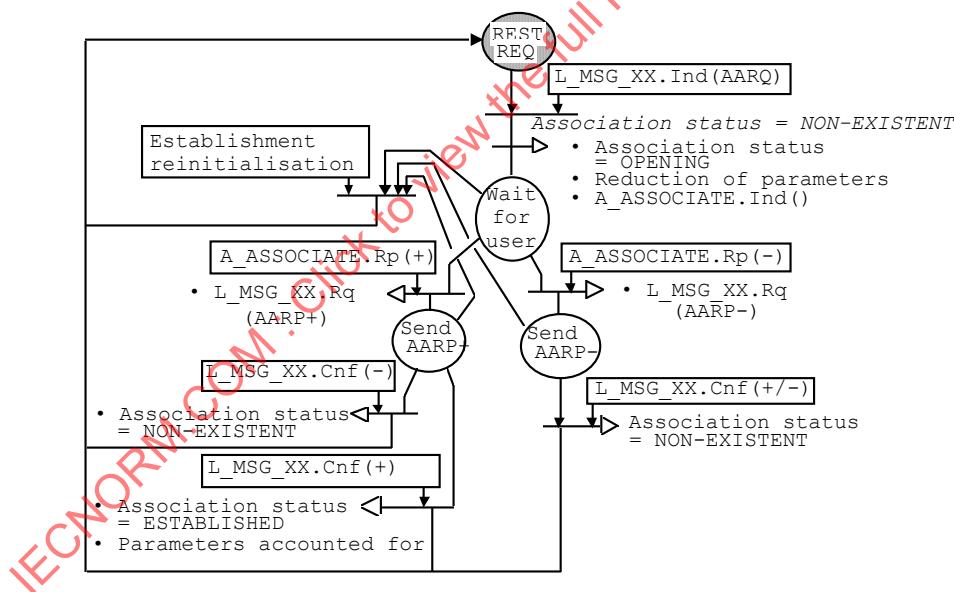


Figure 18 – Association establishment: Responder element state machine

d) Description of mechanism

As soon as the L_MSG_XX.ind() service is invoked with AARPQ-APDU for data, when the association was previously NON-EXISTENT, the association is placed in OPENING status, the A_ASSOCIATE.ind() primitive is invoked, and an quality of service parameter reduction may occur.

The mechanism then enters an A_ASSOCIATE.rp() primitive wait state. As soon as the latter occurs, an AARP, which is positive or negative depending on the establishment response, is sent by invoking the L_MSG_XX.rq primitive.

This link service returns a local confirmation when the message has effectively disappeared from the link queues, after a successful transmission or after a failure.

In the event of failure in transmission of an AARP or success in transmission of a negative AARP signifying establishment refusal, the association status returns to NON-EXISTENT and the resources reserved on receiving the request are freed. The association status changes to ESTABLISHED in the event of successfully transmitting and AARP signifying establishment acceptance.

Finally, as soon as the establishment procedure is begun, i.e. the responder element status is other than “REST REQ”, the latter is reinitialised when the association status changes from OPENING to NON-EXISTENT.

10.2.3 Association termination

10.2.3.1 General

This association termination procedure consists of two parts, one relating to the termination requester and the other to the termination responder.

10.2.3.2 Requester element

a) For the termination requester element, the events and conditions accounted for are the following:

- invocation of the A_RELEASE.rq() primitive by the user,
- reception of an RERP-APDU in the data of the L_MSG_XX.ind() primitive. This PDU contains the “Result” parameter which indicates whether the termination is accepted,
- a “Termination duration” time-out, designed to limit the requester’s wait,
- the invoking of the L_MSG_XX.cnf(+/-) primitive by the data link layer, indicating whether or not the sending has occurred correctly,
- the occurrence of reinitialisation of termination which corresponds to the changing of the association status from CLOSING to ESTABLISHED,
- the association status established value governs the acceptance by the requesting element of the A_RELEASE.rq primitive,
- condition CN = LSNS.

b) The actions carried out by this requester element are the following:

- invoking of the A_RELEASE.cnf(+/-) primitive,
- sending of an RERP-APDU in the data of the L_MSG_XX.rq() primitive,
- setting the time-out which monitors the “Termination duration”, designed to limit the requester’s wait,
- assigning of association status values CLOSING, ESTABLISHED and NON-EXISTENT.

c) Requester element mechanism:

An association in which it is possible to implement the termination procedure supports one single instance of this mechanism at called and calling entity levels. Both ends of the association can request termination.

This mechanism requires not only that all the interactions with the user take place in the context of the association to which the mechanism belongs but also that all interaction with the data link layer take place in the context of the pair of addresses (of ADAEI type) specific to the association. See Figure 19.

NOTE It is important to understand that the A_RELEASE.rq() service invocation request is not necessarily followed by an A_RELEASE.cnf(+/-) confirmation. Indeed, in the case of establishment reinitialisation, it is the A_ABORT.ind() primitive which follows a termination request and thus replaces the confirmation.

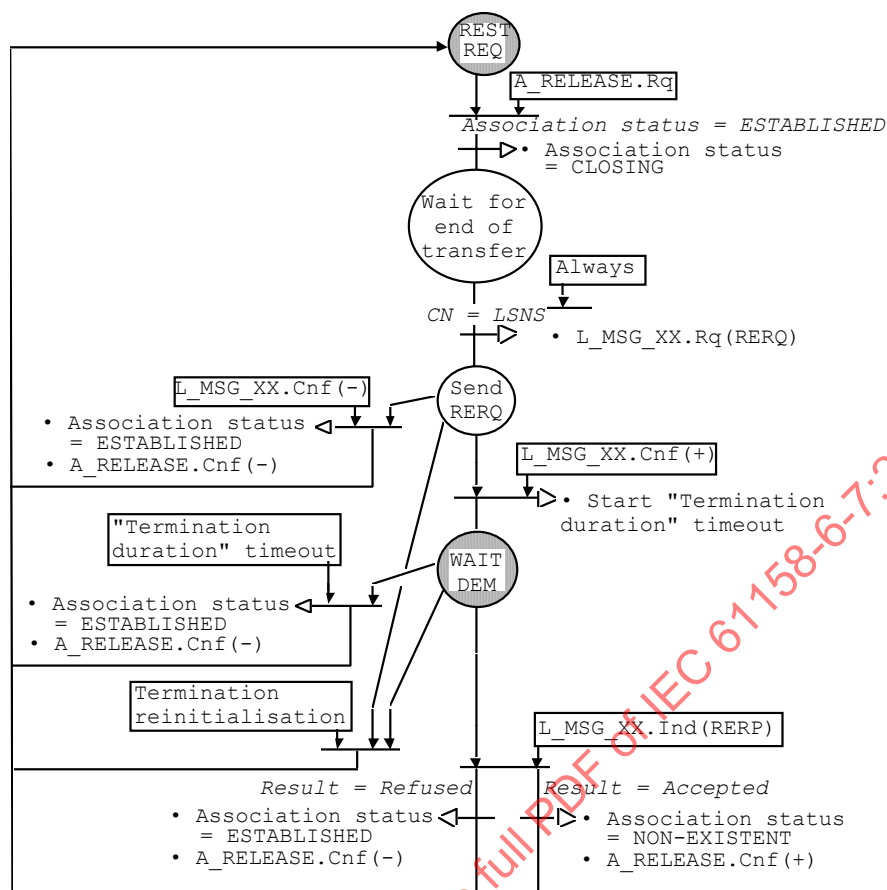


Figure 19 – Association termination: Requester element state machine

d) Description of mechanism:

As soon as the A_RELEASE.rq termination request is carried out in an ESTABLISHED association, its status changes to CLOSING and a PDU is sent requesting termination as soon as all the data transfers in progress in the association are finished (CN=LSNS).

Sending is by invoking the L_MSG_XX.rq() link service which is locally confirmed by L_MSG_XX.cnf() when the message has effectively disappeared from the link queues, subsequent to a successful transmission or subsequent to a failure.

In the case of a link transmission failure, the requester element is reset to quiescent status and the association is set to ESTABLISHED status.

NOTE Termination is a procedure in which priority is given to opening in the event of execution faults.

After transmitting a termination request, the mechanism awaits a termination response, after having started a "Termination duration" time-out.

On the arrival of the termination response, the association status changes to ESTABLISHED or to NON-EXISTENT, depending on whether the termination request was accepted or refused by the responder to this request.

In addition, the wait for a response is interrupted, and the association status becomes ESTABLISHED, if the termination time-out is reached. Finally, as soon as the termination procedure is begun, i.e. the requester element status is "Send RERQ" or "WAIT REQ", the latter is reinitialised with "REST REQ" status when the association changes from CLOSING to NON-EXISTENT.

NOTE This association state transition results from invocation of the ABORT service by the local or remote user or by the local or remote provider element for simultaneous termination purposes.

10.2.3.3 Responder element

a) For the termination responder element, the events and conditions accounted for are the following:

- invoking the A_RELEASE.rp() primitive by the user,
- receiving a RERQ-APDU in the data of the L_MSG_XX.ind() primitive,
- invoking the L_MSG_XX.cnf(+/-) primitive by the data link layer, indicating whether the transmission has taken place properly or not,
- an occurrence of termination reinitialisation, which corresponds to changing the association from the CLOSING to the ESTABLISHED state,
- the association status value CLOSING governs the acceptance of reception of an RERQ by the responder element.

b) The actions carried out by this responder element are the following:

- sending an RERP-APDU in the data of the L_MSG_XX.rq() primitive,
- assigning association status value CLOSING, ESTABLISHED and NON-EXISTENT,
- a provider ABORT request.

c) Responder element mechanism:

An association in which it is possible to implement the termination procedure supports a single instance of the mechanism at calling and called entity levels. Both ends of the association can respond to a termination.

In addition, if both ends request simultaneously, the termination collision is detected by the responder mechanism, which then proceeds to request a provider ABORT for the association.

The mechanism also requires that all interactions with the user be made in the context of the association to which the mechanism belongs and that all interactions with the data link level are in the context of the pair of addresses (of ADAEI type) specific to the association. See Figure 20.



As soon as the L_MSG_XX.ind() service is invoked with an RERQ-APDU for data, when the association is ESTABLISHED, the association state changes to CLOSING and the A_RELEASE.ind() is invoked.

This link service returns a local confirmation when the message has effectively disappeared from the link queues, subsequent to a successful termination or subsequent to a failure.

In all other cases, the status of the association changes to NON-EXISTENT.

Finally, as soon as the termination procedure is begun, i.e. the status of the responder element is other than "REST REQ", the latter is reinitialised when the association status changes from CLOSING to NON-EXISTENT.

10.2.4 Association revocation

10.2.4.1 General

This association revocation procedure consists of two parts: requester and acceptor.

NOTE The difference between the responder and the acceptor is that for the first association revocation is a confirmed service and for the second it is not.

10.2.4.2 Requester element

The requester element is that which causes the sending of an association revocation request to an association.

a) For an association revocation requester element, the events and conditions accounted for are the following:

- invoking the A_ABORT.rq() primitive by the user,
- invoking the L_MSG_XX.cnf(+/-) primitive by the data link layer,
- indicating whether or not sending has taken place correctly,
- the association status values OPENING, ESTABLISHED AND CLOSING govern acceptance of the A_ABORT.rq() primitive by the requester element.

b) The actions which are carried out by this requester element are the following:

- invoking the A_ABORT.ind() primitive,
- sending an ABRQ-APDU in the data of the L_MSG_XX.rq() primitive,
- setting association status value to NON-EXISTENT.

c) Mechanism of requester element:

An association in which it is possible to implement the revocation procedure supports a single instance of this mechanism at called and calling entity level.

This mechanism requires all interaction with the user to take place in the context of the association to which the mechanism belongs and all interaction with the data link level being in the context of the pair of addresses (of ADAEI type) specific to the association. See Figure 21.

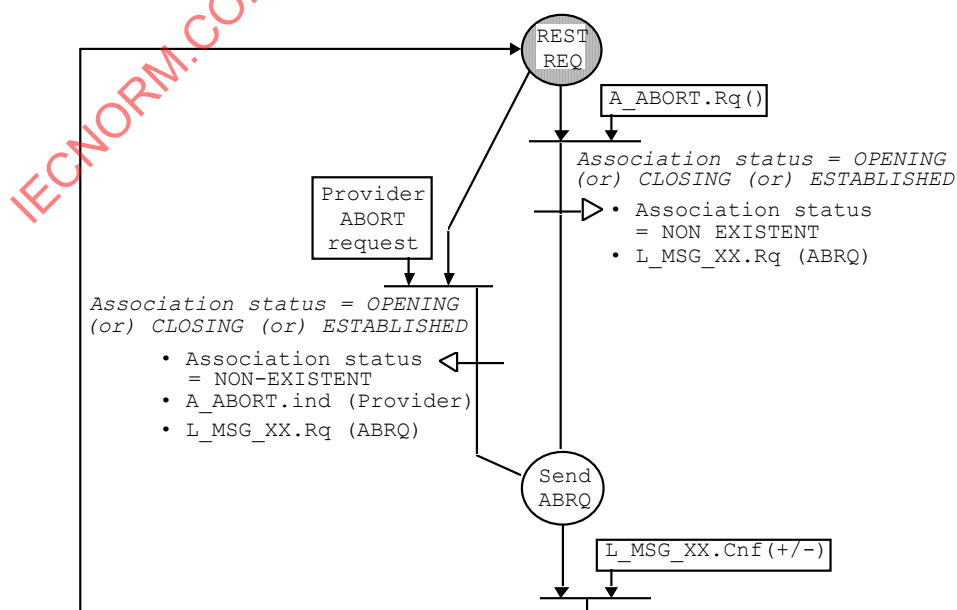


Figure 21 – Association revocation: Requester element state machine

d) Description of mechanism:

As soon as a revocation request from user A_ABORT.rq or the provider is made on an association which is ESTABLISHED, OPENING or CLOSING, the status of the latter changes to NON-EXISTENT and a revocation request PDU is sent. In addition, in the event of a provider attempt, the A_ABORT.ind primitive is invoked to warn the local user.

Whatever the results of sending, all the resources locally reserved for the association are freed.

10.2.4.3 Acceptor element

The acceptor element is that which receives an association revocation request and which processes it regardless of its origin.

a) For the revocation acceptor element, the events and conditions accounted for are the following:

- receiving an ABRQ-APDU in the data of the L_MSG_XX.ind() primitive,
- the OPENING, CLOSING or ESTABLISHED status of the association governs the acceptance by the acceptor element of the receipt of an ABRQ.

b) The actions carried out by this acceptor element are the following:

- invoking the A_ABORT.ind() primitive,
- setting the status of the association to NON-EXISTENT.

c) Acceptor element mechanism:

An association in which it is possible to implement the revocation process, supports a single instance of the mechanism at called and calling entity level.

This mechanism requires all interaction with the user to take place in the context of the association to which the mechanism belongs and for all interaction with the data link layer to be in the context of the pair of addresses (of ADAEI type) specific to the association. See Figure 22.

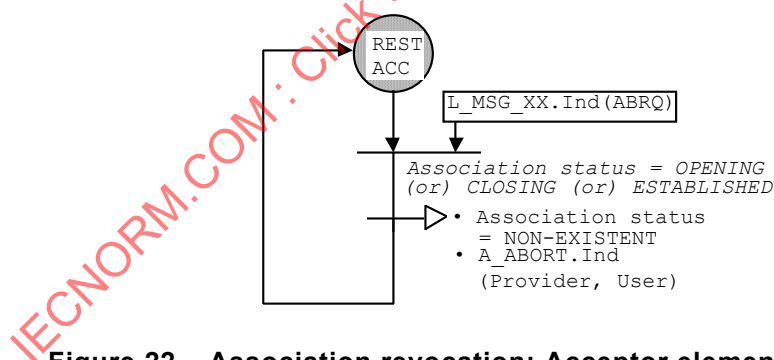


Figure 22 – Association revocation: Acceptor element state machine

NOTE Invocation of the A_ABORT.ind() service of which the origin is the "User" are necessarily from the correspondent, whereas one of "Provider" origin may be local or remote.

d) Description of mechanism:

As soon as the L_MSG_XX() service is invoked with an ABRQ-APDU for data, with the association ESTABLISHED, OPENING or CLOSING, the association status becomes NON-EXISTENT and the A_ABORT.ind() primitive is invoked.

10.2.5 Associated mode data transfer

10.2.5.1 General

In the associated mode, the complexity of the data transfer protocol procedure varies with the supported mechanisms adopted for the association.

The overall procedure is thus composed of a number of state machines respectively associated with the various mechanisms which can be implemented in association, interacting via calls and responses or via global variables.

The various state machines defined for the mechanisms which can be implemented in an association at data transfer requester or acceptor level are the following:

- TS_SM_RQ, TS_SM_ACC:
Transfer service machine for requester and acceptor.
- AT_SM_RQ, AT_SM_ACC:
Non-acknowledged transfer machine for the requester and the acceptor.
- ATA_SM_RQ, ATA_SM_ACC:
Acknowledged transfer machine for the requester and the acceptor.
- NB_SM:
Numbering machine at requester level.
- RC_SM:
Retry machine at requester level.
- DU_SM:
Antiduplication machine at acceptor level.
- MA_SM:
Anticipation machine at requester level.
- SG_SM:
Segmentation machine at requester level.
- RS_SM:
Reassembly machine at acceptor level.

NOTE The term “acceptor” is used for the receiver of a data transfer, as it is an unconfirmed service. For a confirmed service, the term “responder” would be used.

All state machines interact both:

- between themselves in the context of an association. These interactions take place by MCS local identification of calls/responses between machines,
- with other application service elements participating at the same association. These take place by local identification between ASEs,
- with the data link layer in the case of a pair of link addresses of the ADAEI type, which localises the ends of the association.

Figure 23 shows interaction between the various state machines.

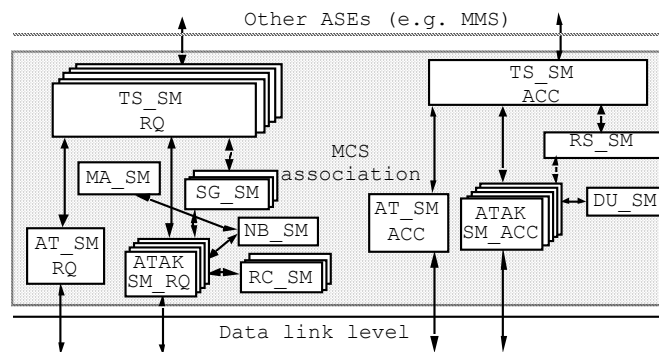


Figure 23 – Interactions between state machine in an associated mode data transfer

This breakdown of the associated mode data transfer procedure uses modularity associated with the classes of conformity. It is thus that the only necessary machine is that of the transfer service which, in conjunction with the acknowledged or unacknowledged transfer machine, constitutes the simplest classes of conformity for transfer of data in the associated mode.

10.2.5.2 Definitions

10.2.5.2.1 Variables

The various state machines implemented in the context of an association for the transfer of data in the associated mode are co-ordinated by variables. There are two types of variables:

- local variables, which are specific to one status machine and which have a value which exists and may vary in the context of each of the invocations of the machine,
- global variables, which makes it possible to manage interaction between the different mechanisms and which have a value which exists and possibly varies in the context of each association.

The local variables are described at the level of each of the mechanisms to which they belong, and the global variables are described below.

SA (Service Access): Access to the service used to specify whether a new transfer service can be invoked in the association. (TS_SM_RQ), in order to avoid interlacing of the data transfers associated with a number of service invocations. This variable is shared by TS_SM_RQ and SEG_SM machines.

The value varies dynamically between:

- FREE if another invocation is possible,
- BUSY if it is temporarily impossible to invoke another transfer service.

Typically, the status of the variable changes from FREE to BUSY when the transfer service machine, on the requester side, activates a segmentation machine.

This it remains in the BUSY state for the time interval during which the segmentation machine divides the messages into packets and instantiates as many ATAK_SM_RQ acknowledged transfer protocol machines as there are packets. The status of the SA variable changes from BUSY to FREE in two cases:

- when all the packets subsequent to the message have been invoked by an ATAK_SM_RQ machine or
- if the service is interrupted after negative confirmation of a packet (other than the last one) originating from the ATAK_SM_RQ machine invoked for the packet. Indeed, in this case the message is considered to have been corrupted and the current transfer is interrupted, whereafter another transfer can be envisaged.

AC (Anticipation Credit): The anticipation credit is utilised to specify that another acknowledged transfer (ATAK_SM_RQ) can be invoked in the association.

Another transfer can only be invoked within the quantitative limits set by the anticipation factor provided that the difference between the sequence number of the earliest invocation and the current number is less than the value of the anticipation factor.

The value varies dynamically between:

- FREE if a new invocation is possible,
- BUSY if it is temporarily impossible to invoke another acknowledged transfer.

On opening an association, the variable is initialised at FREE.

The mechanism controlling variation of the value of this variable is MA_SM.

CN (Current Number): The sequence number to be allocated to the next acknowledged data transfer.

This variable is shared by the machines NB_SM, MA_SM, SEG_SM and ATAK_SM_RQ.

The value varies under the control of the numbering mechanism (NB_SM).

This variable is initialised at “0” on opening the association.

LSNS (Lowest Sequence Number Send): The lowest sequence number attributed to an acknowledge transfer being exchanges and known at transfer initiator level.

The sequence numbers (SN) identify the invocations of transfer in the associated mode. Each of the PDUs sent contains a sequence number, which is necessary for numerous mechanisms.

This variable is initialised at “0” on opening the association.

S (Size): Size of service expressed in terms of the number of packets of maximum size.

a (Anticipation factor): This is the anticipation factor adopted for the association. The minimum value of **a** is 1.

RC: Specifies whether retry (RC_SM) is adopted in the association. The value of this variable is static throughout the life of an association and is governed by the fact that the two devices participating in it belong to a class of conformity which supports it.

The values are the following:

- USED: mechanism adopted, necessarily being supported by the devices,
- UNUSED: mechanism not adopted as not supported, or not supported by any one of the devices.

DU: Specifies whether antiduplication (DU_SM) is adopted in the association. The value of this variable is static throughout the life of an association and is governed by the fact that the two devices participating in it belong to a class of conformity which supports it.

The values are the following:

- USED: mechanism adopted, necessarily being supported by the devices,

- **UNUSED:** mechanism not adopted as not supported, or not supported by any one of the devices.

SG: Specifies whether segmentation (SG_SM) is adopted in the association. The value of this variable is static throughout the life of an association and is governed by the fact that the two devices participating in it belong to a class of conformity which supports it.

The values are the following:

- **USED:** mechanism adopted, necessarily being supported by the devices,
- **UNUSED:** mechanism not adopted as not supported, or not supported by any one of the devices.

RS: Specifies whether reassembly (RS_SM) is adopted in the association. The value of this variable is static throughout the life of an association and is governed by the fact that the two devices participating in it belong to a class of conformity which supports it.

The values are the following:

- **USED:** mechanism adopted, necessarily being supported by the devices,
- **UNUSED:** mechanism not adopted as not supported, or not supported by any one of the devices.

10.2.5.2.2 Operators

The different state machines have conditional transitions which are expressed with arithmetic in $2^{\text{exp}16}$ modulo numbers. Indeed, all MCS-APDU identification is carried out with 16-bit binary-coded numbers which vary between 0 and $2^{\text{exp}16}-1 = \text{Max}$.

Consequently, to facilitate expression of conditions, operators have been defined:

- **“successor”**
When $y = \text{success}(x)$; $y = x+1$, if $x < \text{Max}$; and $y = 0$, if $x = \text{Max}$
- **“distance”**
When $y = \text{dist}(a,b)$; $y = a-b$, if $a \geq b$; and $y = a + \text{max} + 1 - b$, if $a < b$
- **“between”**
When $(x) \text{between}(a,b) = y$;
 $y = \text{TRUE}$, if $b > a$ and $x \geq a$ and $x < b$,
 if $a > b$ and $x \geq a$,
 if $a > b$ and $x < b$
 $y = \text{FALSE}$, in all other cases.

NOTE In the special case where a is equal to b , y takes the value FALSE.

10.2.5.3 Transfer service (TS_SM)

10.2.5.3.1 General

The transfer service machine consists of two parts, one relating to the requester and the other to the acceptor.

10.2.5.3.2 Requester element (TS_SM_RQ)

The requester element is that which causes the transmission of a data transfer request on an association. See Figure 24.

- a) For the transfer service request element, the following events and conditions are accounted for:

A_DATA.rq(Request_ack), where Request_ack = TRUE (or) FALSE

dt_noack_cnf(+/-) (see 10.2.5.4.2)

dt_ack_cnf(+/-) (see 10.2.5.5)

- segment_cnf(+/-) (see 10.2.5.9)
- Association status = ESTABLISHED
- AC = FREE
- SA = FREE
- SG = USED or UNUSED.

- b) The actions carried out by the requester element are the following:

A_DATA.cnf(+/-)

- dt_noack_rq() ,
- dt_ack_rq(Packet nature), where Packet nature = NIL (see 10.2.5.2.1),
- segment_rq() (see 10.2.5.9).

- c) Mechanism of requester element:

An association necessarily has a transfer service mechanism and can support one or more instances at called and calling entity level. The multiplicity of instances is limited by the maximum anticipation value specified in an association. Indeed, the number of instances possible is equal to the maximum anticipation value if there is no segmentation, and less than or equal to it otherwise.

NOTE If the segmentation mechanism is adopted, the number of instances of the mechanism possible in an association is equal, on average, to maximum anticipation "a" divided by the average number of packets necessary for the exchange of a service.

Machine: TS_SM_RQ

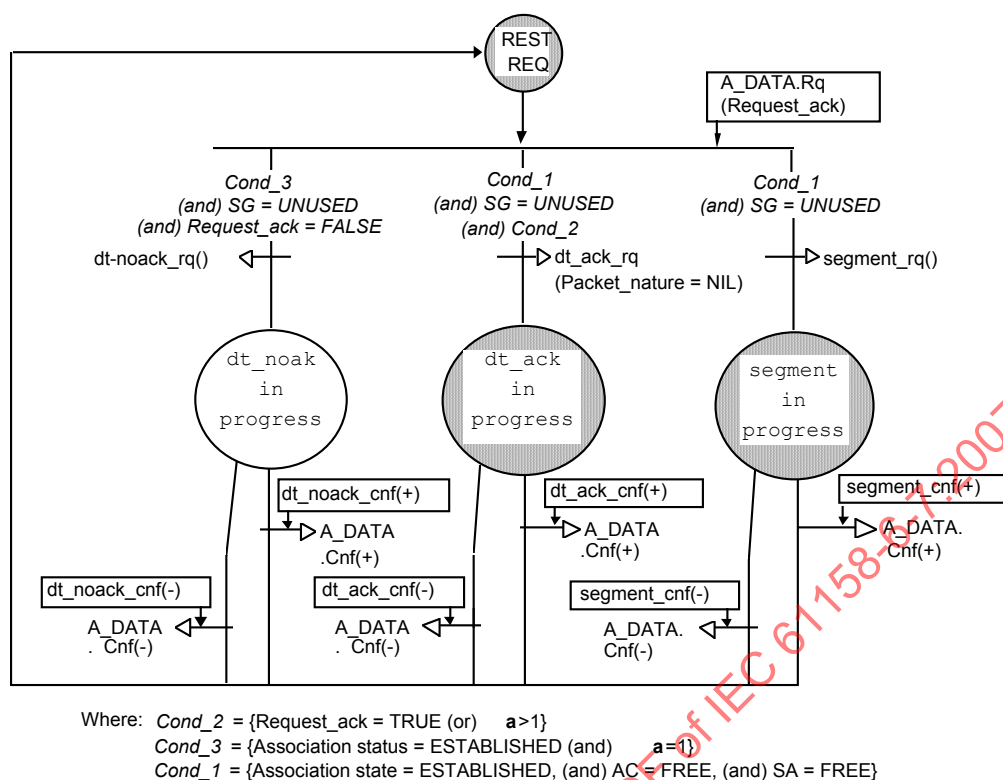


Figure 24 – Transfer service: Requester element state machine

d) Description of mechanism:

As soon as a `A_DATA.rq(Request_ack)` data transfer request is made by the user, a call to another mechanism is made depending on the value of the global variables of the association and the "Request_ack" parameter transferred in the request.

Thus, an acknowledged transfer, or an unacknowledged transfer or a segmentation are begun.

In the case of an association which implements segmentation, the "Request_ack" parameter can only have the value TRUE.

10.2.5.3.3 Acceptor element (TS_SM_ACK)

The acceptor element is that which receives a data transfer in an association.

a) For the transfer service acceptor element, the following events and conditions are accounted for:

- `dt_noack_ind()` (see 10.2.5.4.3)
- `dt_ack_ind()` (see 10.2.5.5.3)
- `reassemb_ind()` (see 10.2.5.10)
- Association status = ESTABLISHED (or) CLOSING
- RS = USED (or) UNUSED

b) The actions carried out by this acceptor element are the following:

- `A_DATA.ind(Request_ack)`, where `Request_ack` = TRUE, (or) FALSE

NOTE This parameter necessarily has the value TRUE if reassembly takes place.

c)

Acceptor element mechanism (see Figure 25):

An association necessarily has a transfer service mechanism and may support a number of instances of the machine for local anticipation purposes at device level.

Machine: TS_SM_ACC

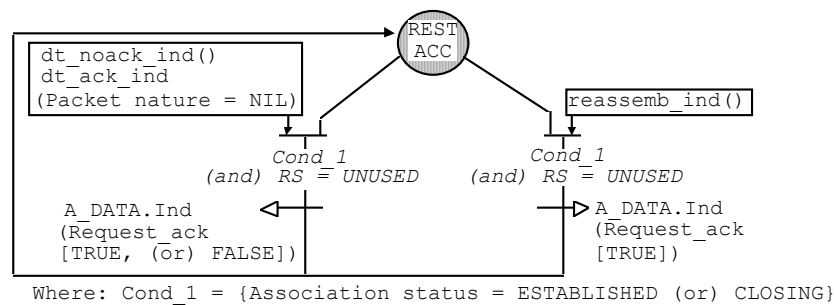


Figure 25 – Transfer service: Acceptor element state machine

d) Description of mechanism:

As soon as another RS_SM or AT_SM_ACC, or ATAK_SM_ACC machine signals receiving the data transfer, this indication is accounted for depending on the value of the global variables of the association.

Its accounting for involves returning an A_DATA.ind(Request_ack) data transfer indication to the user. In the case where an association implements reassembly, the parameter "Request_ack" can only take the value TRUE.

10.2.5.4 Unacknowledged transfer (AT_SM)

10.2.5.4.1 General

In unacknowledged associated mode data transfer, the machine consists of two parts, one relating to the requester and the other to the acceptor.

10.2.5.4.2 Requester element (AT_SM_RQ)

The requester element is that which causes sending of a data transfer request in an association.

a) For the unacknowledged transfer requester element, the following events and conditions are accounted for:

- dt_noack_rq(),
- L_MSG_XX.cnf(+/-) (see IEC 61158-3-7, Type 7).

b) The actions carried out by this requester element are the following:

- L_MSG_XX.rq(ATRQ[noack]) (see IEC 61158-3-7, Type 7),
- dt_noack.cnf(+/-).

c) Requester element mechanism (see Figure 26):

An association in which it is possible to implement the non-acknowledged data transfer machine may support a number of instances of this machine for local anticipation purposes at device level.

Machine: AT_SM_RQ

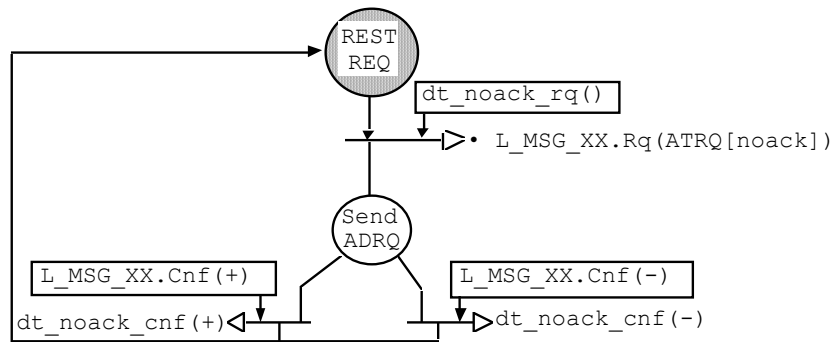


Figure 26 – Unacknowledged transfer: Requester element state machine

d) Description of mechanism:

As soon as a call is made to this machine with `dt_noack_rq`, an ATRQ data transfer which does not contain an acknowledgement request is sent.

Whatever the results of sending, the mechanism returns to its initial status.

NOTE Sending errors may be accounted for by equipment fault management.

10.2.5.4.3 Acceptor element (AT_SM_ACC)

The acceptor element is that which receives a data transfer with no reception acknowledgement requests and which transfers it to the user.

a) For this element, the following events and conditions are accounted for:

`L_MSG_XX.ind(ATRQ[noack])` (see IEC 61158-3-7, Type 7).

b) The following actions are carried out by this acceptor element:

- `dt_noack_ind()`

c) Acceptor element mechanism (see Figure 27):

An association in which it is possible to implement the non-acknowledged data transfer procedure may support a number of instances of this machine for local anticipation purposes at device level.

Machine: AT_SM_ACC

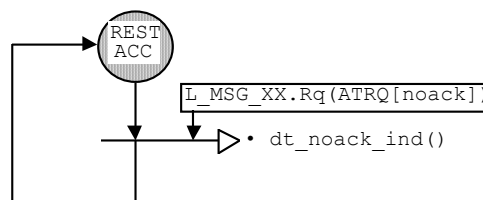


Figure 27 – Unacknowledged transfer: Acceptor element state machine

d) Description of mechanism:

As soon as an `L_MSG_XX.ind()` link service is invoked, the data of which correspond to an ATRQ without an acknowledgement request, a `dt_noack_ind` signal is given.

10.2.5.5 Acknowledged transfer (ATAK_SM)

10.2.5.5.1 General

The acknowledged associated mode data transfer machine consists of two parts, one relating to the requester and the other to the acceptor.

10.2.5.5.2 Requester element (ATAK_SM_RQ)

The requester element is that which causes transmission of a data transfer request and receives the corresponding acknowledgement in an association. See Figure 28.

a) For this requester element, the events and conditions accounted for are the following:

- dt_ack_rq(Packet nature), where Packet nature = STT, (or) MID, (or) END, (or) ALL, (or) NIL.

Packet nature NIL is used when this requester element is invoked outside segmentation.

- L_MSG_XX.cnf(+/-) (see IEC 61158-3-7, Type 7)
- L_MSG_XX.ind(ATAK[Packet_num])
- acknowledgement time-out
- retry_cnf(+/-) (see 10.2.5.6)
- revocation AT_SM_RQ
- RC = USED (or) UNUSED
- SN = Packet_num
- (SN) between (FSN,LSN) = TRUE

b) The following actions are carried out by this requester element:

- SN = Take number (see 10.2.5.5.4)
- L_MSG_XX.rq(ATRQ[ack]) (see IEC 61158-3-7, Type 7)
- resetting of acknowledgement time-out
- retry_rq() (see 10.2.5.6)
- dt_ack_cnf(+/-)
- free(SN)
- retry revocation

c) Request element mechanism:

For an association in which it is possible to implement the data transfer mechanism with acknowledgement, the number of instances of the machine which can be implemented simultaneously both at calling entity and called entity level is linked to the value of the anticipation factor and the value of the anticipation credit. The anticipation can be implemented simultaneously both at calling entity and called entity level is linked to the value of the anticipation factor and the value of the anticipation credit. The anticipation factor determines the greatest possible number and the credit establishes the greatest instantaneous number on the basis of constraints associated with the retry and antiduplication mechanism designed to increase transmission reliability.

Machine: ATAK_SM_RQ

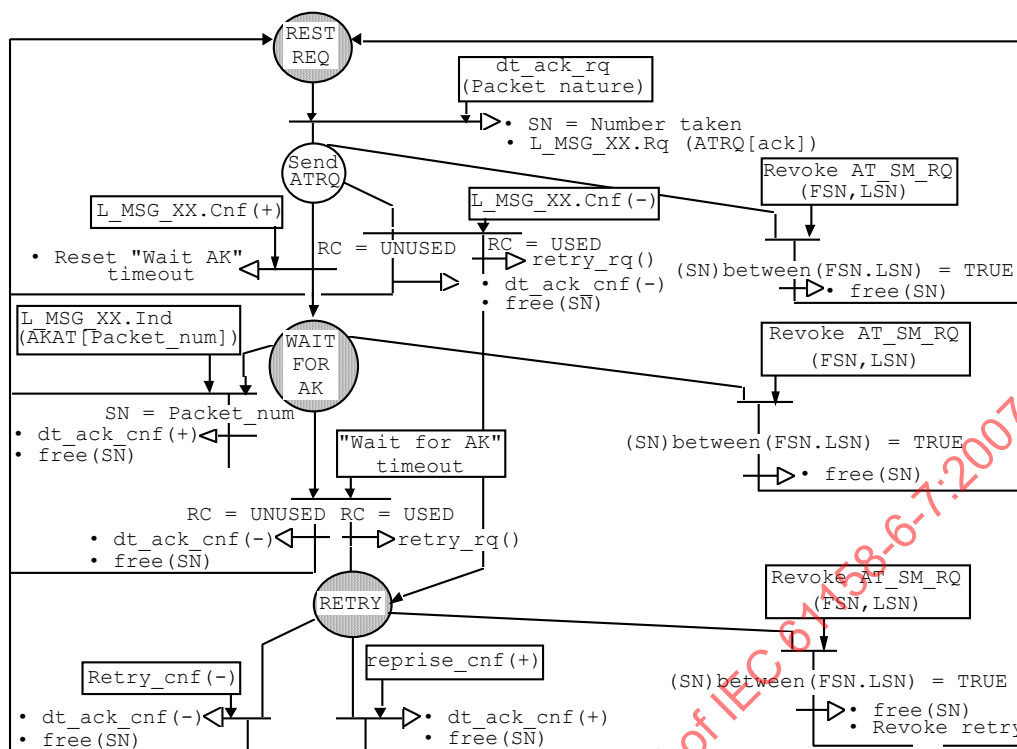


Figure 28 – Acknowledged transfer: Requester element state machine

d) Description of mechanism:

As soon as the unacknowledged data transfer request is made, an ATRQ containing an acknowledgement request is sent.

The “Packet nature” parameter contains values STT, MID, END and ALL if filled in by the segmentation machine and NIL otherwise. This parameter is coded in the ATRQ.

If the result of sending return by the data link layer is positive, an acknowledgement time-out is begun. The state is one of waiting for acknowledgement, which either arrives, in which case the mechanism returns to its initial state, or the acknowledgement time-out expires and a retry may take place before returning to the same initial status.

Furthermore, if after a transfer request, the data link level returns a negative result, a retry is immediately requested if the latter is utilised.

Any invoking of this machine involves taking a sequence number from the global numbering machine of the association. This is stored in the “SN” variable throughout its lifetime. Finally, any event leading to the revocation of an invocation, whether completed normally or abnormally, results in freeing of the sequence number to manage the value of the lowest sequence number (LSNS) still current in the association.

Finally, as soon as an instantiation of this mechanism is in another state than “REST-REQ”, it can be reinitialised at a “Revocation AT_SM_RQ” request from the segmentation mechanism.

10.2.5.5.3 Acceptor element (ATAK_SM_ACC)

The acceptor element is that which receives the data transfer and the acknowledgement in an association. See Figure 29.

a) For this acceptor element, the following events and conditions are taken into account:

- L_MSG_XX.ind(ATRQ[ack]) (see IEC 61158-3-7, Type 7)

antiduplication_cnf(Message), where Message = REPETITION, (or) NEW

- L_MSG_XX.cnf(+/-) (see IEC 61158-3-7, Type 7)
- DU = USED, (or) UNUSED

b) The actions carried out by this acceptor element are the following:

- antiduplication_rq(Sequence number), Sequence number has the value derived from ATRQ[ack]
- L_MSG_XX.rq(AKAT) (see IEC 61158-3-7, Type 7)
- dt_ack_ind(Sequence number; Packet nature) where Packet nature has the value derived from the ATRQ received and Packet nature = STT, (or) MID, (or) END, (or) ALL, (or) NIL (see 10.2.5.5.2).

c) Acceptor element mechanism:

An association in which it is possible to implement the data transfer procedure may support a number of instances of this machine for local anticipation purposes at device level.

Machine: ATA_K_SM_ACC

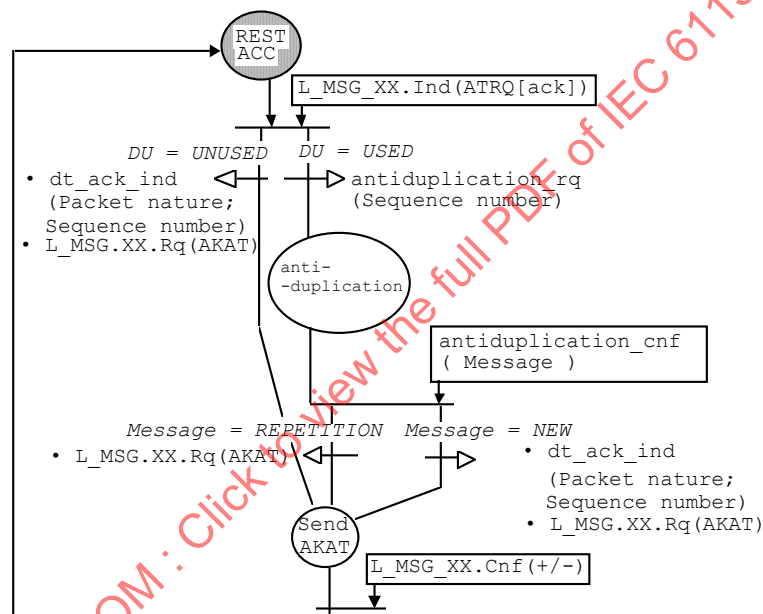


Figure 29 – Acknowledged transfer: Acceptor element state machine

d) Description of mechanism:

As soon as the L_MSG_XX.ind primitive is invoked, with data which corresponds to an ATRQ with acknowledgement request, a condition is evaluated before proceeding with sending an acknowledgement, and return of the message. This condition concerns the fact that antiduplication is implemented in the association.

If this is adopted, depending on whether it considers the message as NEW or as a REPETITION, it is or is not returned.

In all cases, an acknowledgement is returned to the data transfer sender, containing a sequence number which is equal to that of the data transfer acknowledged.

Finally, whether or not sending of the transmission fails at link level, the mechanism returns to its initial state.

10.2.5.5.4 Numbering (NB_SM)

On using the data transfer service with acknowledgement in an association, the numbering element is implemented to manage allocation of sequence numbers (SN). The values of this number are transited together with the data in the transfers using the ATRQ type PDU, and

which are also found in the respectively associated acknowledgements, of which the PDU is of the AKAT type. See Figure 30.

In addition, the state machines implemented at requester and acceptor levels, for acknowledged data transfers, retain this number during their lifetime to make them identifiable by the protocol in the context of the association.

Furthermore, this machine manages freeing of the sequence numbers. Freeing takes place each time a data transfer with acknowledgement terminates normally or abnormally. This management system makes it possible to update the value of the LSNS variable.

a) This mechanism accounts for the following events and conditions:

- “Take number”
- A_ASSOCIATE.rp(+), (or) A_ASSOCIATE.cnf(+)
- A_RELEASE.rp(+), or A_RELEASE.cnf(+), or A_ABORT.ind()
- free(SN).

b) The actions triggered by this mechanism are the following:

- CN = success(CN)
- LSNS management
- ma_rq()
- LSNS = CN = 0
- LACT[i] = Not received, when $0 \leq i \leq \text{Max}$.

c) Mechanism:

There is a single instance of this mechanism per association at calling entity level and at called entity level. This mechanism shall necessarily exist at the level of one association if the latter supports data transfer with acknowledgement.

Machine: NB_SM

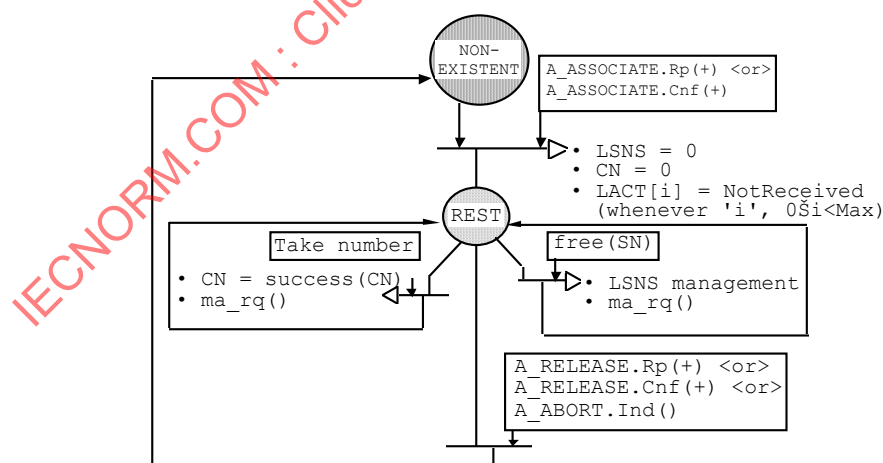


Figure 30 – Numbering mechanism state machine

d)

Description of mechanism:

A numbering entity (NB_SM) is instantiated and utilised on opening an association. Whenever the “Take number” function is invoked by another mechanism causing reading of the value followed by incrementation. The value taken is used to identify an

acknowledged data transfer. In addition, this value circulates in the acknowledged data transfer PDUs but also in the associated acknowledgement.

Whenever the free function (SN) is invoked by another mechanism management of the LSNS variable takes place. Management of the LSNS is based on a list LACT[i] containing “Max” elements (see 10.2.5.2.2) and proceeds to the following operations:

- if SN≠LSNS, then account for the reception by setting LACT[SN]=Received,
- if SN=LSNS, then repeat LSNS=success(LSNS) and LACT[SN]=Not received until LACT[LSNS]≠Not received.

10.2.5.6 Retry (RC_SM)

On using the data transfer service with acknowledgement in an association, the retry element can be implemented if adopted in the association. See Figure 31.

This element can only be implemented conjointly with that of antiduplication (10.2.5.7).

This element makes it possible to reiterate, a number of times, a data transfer with acknowledgement in the event of absence of receiving an acknowledgement after a specified time limit has elapsed.

a) The following events and conditions are accounted for by this mechanism:

- retry_rq()
- Acknowledgement time-out
- Counter: retry counter
- Rpmx: maximum value of retry counter adopted for the association
- L_MSG_XX.cnf(+/-) (see IEC 61158-3-7, Type 7)
- L_MSG_XX.ind(AKAT) (see IEC 61158-3-7, Type 7)
- Retry revocation.

b) The following actions are triggered by this mechanism:

- retry.cnf(+/-)
- L_MSG_XX.rq(ATRQ) (see IEC 61158-3-7, Type 7)
- Incrementation of counter
- Starting of “Wait for AK” time-out.

c) Mechanism:

There is one instance of this mechanism per request for data transfer with acknowledgement being retried. Each of these mechanisms has its own current counter value (Counter). As for the maximum value of the retry counter (Rpmx), it is that which is specific to the association in which it is implemented.

Machine: RC_SM

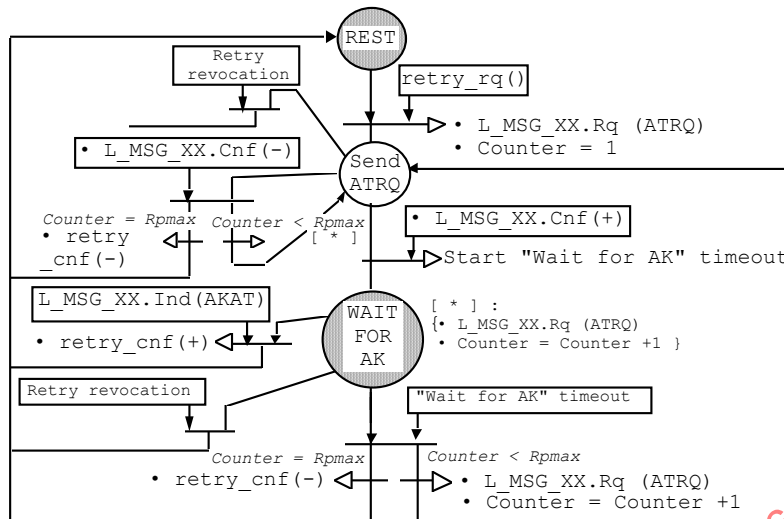


Figure 31 – Retry mechanism state machine

d) Description of mechanism:

This mechanism simply proceeds with resending a data transfer within the limit of the maximum number set per association, knowing that a first transmission has already taken place.

Resending takes place in the event of negative confirmation by the data link layer or in the event of positive confirmation if the acknowledgement arrives after the time interval during which it is expected.

10.2.5.7 Antiduplication (DU_SM)

During data transfers with acknowledgement in an association, the antiduplication element is implemented if adopted in the association. It is used conjointly with the retry element (see 10.2.5.6).

This element makes it possible to detect, at acceptor level, an ATRQ which has already been the subject of a data transfer with acknowledgement, as a number of repetitions of the same ATRQ can occur after a transmission fault.

a) The following events are accounted for by this mechanism:

- antiduplication_rq(Sequence number)

NOTE The acknowledged data transfer acceptor mechanism which invokes this mechanism transmits the sequence number available to the ATRQ for which it wants to make an antiduplication check.

b) The following actions are triggered by this mechanism:

- antiduplication_cnf(Message), where Message = REPETITION or NEW.

c) Mechanism:

There is a single instance of this mechanism per association at calling entity and called entity level.

This antiduplication mechanism is purely transformational and there is therefore no state machine but simply a description of the data and processing associated with it.

- Data description:

The processing is based on an “antiduplication list” containing information concerning the latest data transfers with acknowledgement which have taken place.

This antiduplication list is an LAD[i] structure table in which the number of elements is equal to the anticipation factor adopted for the association, and of which each structure features a sequence number and reception status associated with the transfer LAD[i] = (sn[i], Etr[i]).

At any given moment, the first element in the table has a sequence number that was the greatest (HSNA) of those received in the ATRQs for which the antiduplication mechanism was invoked. Consequently, this first element necessarily has reception status value YES.

The other elements contain successively smaller sequence numbers and a reception status with a value of YES or NO depending on whether the antiduplication mechanism has been invoked or not.

NOTE It is important to understand that the value of the sequence number allocated by the numbering element increases as modulo $2^{\exp(n)}$ where $n=16$, in view of 16-bit representation of the sequence number in the PDUs. See Table 25.

Table 25 – Structure of the antiduplication list

Sequence number	Reception status
sn(1)	YES
sn(2) where, $\text{dist}(\text{sn}(1), \text{sn}(2))=1$	YES/NO
sn(3) where, $\text{dist}(\text{sn}(1), \text{sn}(3))=2$	YES/NO
sn(a) where, $\text{dist}(\text{sn}(1), \text{sn}(a))=a-1$	YES/NO

- Processing description:

When the mechanism is invoked with $\text{antiduplication_rq}(\text{Sequence number})$, the value of the parameter SN sequence number governs the processing:

- When sn belongs to the antiduplication list, the nature of the processing in accordance with the value of the reception status associated with it in the list is as follows:
- if reception status is NO, it is set to YES and the message is considered as NEW,
- if reception status is YES, it stays at YES and the message is considered as a REPETITION.
- When it does not belong to the antiduplication list, it is considered to be new and the processing consists in offsetting the elements in the antiduplication list so that the value of sn(1) corresponds to the received SN sequence number.

The processing consists of three stages, after assessment of the distance $N = \text{dist}(\text{SN}, \text{sn}(1))$.

- if $N < a$, then for the elements in the LAD[i] list, where i decreases from a to $N+1$, make $\text{LAD}[i] = \text{LAD}[i-N]$. This constitutes shifting the elements of the structure by N cells.
- when i increases from 2 to $\text{Min}(N, a)$, make $\text{sn}[i] = M$ so that $\text{dist}(\text{SN}, M) = i-1$ and $\text{Etr}[i] = \text{NO}$. It is a question of setting to not received the reception status associated with the sequence numbers contained exclusively within the values of sn(1) and SN.
- make $\text{sn}(1) = \text{SN}$, and $\text{Etr}(1) = \text{YES}$. It is a question of setting the first element of the list to received.

10.2.5.8 Anticipation (MA_SM)

10.2.5.8.1 General

During transfer of data with acknowledgement in an association, the anticipation element is implemented if adopted in the association.

This element monitors the traffic in the association.

10.2.5.8.2 Requirements

Anticipation relates to transfers of data with acknowledgement in an association. It makes it possible to envisage commitment of a new acknowledged transfer when the acknowledgement of other current transfers has not yet arrived. This makes it possible to optimise the pass-band by multiplexing the data transfers and acknowledgements on the bus.

The anticipation possibilities offered by an association are linked both to the anticipation factor which characterises the association and to the anticipation credit which may temporarily reduce them.

10.2.5.8.3 Anticipation factor

Anticipation factor “a” corresponds to the maximum number of transfers which can be awaiting acknowledgement.

The anticipation factor “a” specific to each direction of transmission in an association establishes both:

- the maximum number of acceptor and requester elements for data transfer with acknowledgement which can be invoked simultaneously,
- and the size of the antiduplication list necessary at the acceptor in the event of implementation of retry and antiduplication.

10.2.5.8.4 Anticipation credit

The anticipation credit which is specific to each end of the association, establishes its capacity whether or not to allow invocation of new requester elements of data transfer with acknowledgement within the limits of the anticipation factor. See Figure 32.

The anticipation credit (AC) is characterised by two values:

- FREE, if it is possible to invoke an acknowledged data transfer requester element,
- BUSY, if it is temporarily impossible to invoke a requester element.

Anticipation credit changes between states depending on the following events:

- ma_rq()

Anticipation credit changes make allowance for two other variables:

- LSNS: the lowest sequence number attributed to an ATRQ sent and awaiting acknowledgement (Lowest Sequence Number Send),
- CN: the current value of the numbering entity for transfer of data with acknowledgement.

Machine: MA_SM

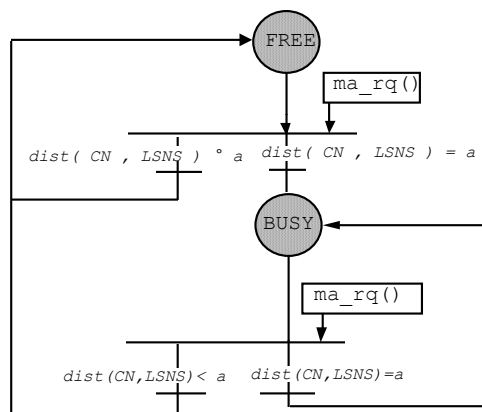


Figure 32 – Anticipation mechanism state machine

10.2.5.8.5 Optimum performance

For the anticipation factor adopted for a given association to constitute the optimum as regards the traffic adopted for it, it shall allow for the following factors:

- ensuring best use of the traffic adopted for the association.

NOTE 1 If it is too large, the maximum number of instances will never be reached, and if it is too low the pass-band will be wasted.

- ensure that the time necessary for chaining the maximum number of retries will be close to that necessary for passing the anticipation traffic.

NOTE 2 This ensures that the data transfer requester user will not be unable to access the association due to using up the anticipation credit by the current repetitions.

As for the user of the acknowledged data transfer service, it will choose an anticipation factor and a traffic which corresponds to these requirements in terms of:

- the capacity to relay a number of data transfer requests (A_DATA) in a given time.

NOTE 3 In the MMS environment, this capacity integrates both the “Outstanding” services and the time taken for execution of the services by the VMD.

- data transfer size, which in the case of segmentation implements a number of MCS transfers for different packets.

10.2.5.9 Segmentation (SG_SM)

A segmentation element can be implemented on an association. This element makes it possible to divide the data unit (SDU) associated with a data transfer service into one or more packets depending on its size, and to manage the acknowledged exchange of the different packets. The implementation of this element at data transfer initiator level is only possible conjointly with that of reassembly at receptor level. See Figure 33.

Global variables utilised by this mechanism:

- AC (Anticipation Credit), global for all association mechanisms,
- SA (Service Access) global for all association mechanisms,
- LSNS (Last Smallest Sequence Number), the smaller sequence number attributed to a transfer currently exchanged.

Local variables of the mechanism:

- PN (Packet Number), the number of packets of an SDU remaining to be sent,
- LSN (Last Sequence Number), the sequence number of the next lowest value, in the modulo sense, to that attributed to the latest packet.

- FSN (First Sequence Number), the sequence number attributed to the first packet arising from segmentation,
- Count_ack, the received acknowledgement counter, relating to transfer of packets of a service.

a) This mechanism accounts for the following events and conditions:

- segment_rq(),
- “End of division (n)”, resulting from the division operation, where parameter (n) is the number of packets obtained,
- Always, permanent event to indicate that the transition is receptive at all times,
- dt_ack_cnf(+/-),
- AC = FREE, to account for the end of sending of packets belonging to the same SDU, or otherwise to indicate whether the packet is the last or not,
- $PN > 1$; $PN = 1$; $PN = 0$; to test for the end of sending of packages belonging to the same SDU, or otherwise to indicate whether the packet is the last or not,
- Count_ack = n, to verify that all the acknowledgements associated with the packets of a service have been received.

b) The following actions are triggered by this mechanism:

- Setting SA to BUSY or FREE,
- “SDU division request”,
- $PN = n$; initialisation of PN at value n of the number of packets resulting from division of the SDU,
- Packet nature = STT; for the first packet of a segmentation, or ALL if the number of packets arising from subdivision is 1,
- $FSN = CN$ and $LSN = x$ so that $dist(x, CN) = n$; initialisation of the variables FSN and LSN,
- Count_ack = 0, initialisation at zero of the acknowledgement counter,
- $Count_ack = Count_ack + 1$, incrementation of the acknowledgement counter,
- dt_ack_rq(Packet nature), where Packet nature = MID, END,
- $PN = PN - 1$; decrementation of the number of packets remained to be sent,
- Packet number = MID or END, depending on whether it is the last packet or not,
- segment_cnf(+/-),
- AT_SM_RQ revocation; revocation of all ATAK_SM_RQ machines of which the associated sequence number is between LSN (value excluded) and FSN (value included). It being impossible to transmit one of these packets, the operation becomes globally impossible and results in interruption of transmission of the packets not yet sent. Thus, the lowest sequence number during transmission (LSNS), will necessarily change to a value equal or greater than LSN.

NOTE 1 Revocation of the ATAK_SM_RQ is atomic with regard to the lifetime of all the machines of the association.

c) Mechanism:

For an association in which it is possible to implement segmentation, the number of instances is bounded by the maximum anticipation value specified in the association. Indeed, the number of possible instances is equal to the maximum anticipation if the SDUs exchanged only call a single packet, and otherwise less.

NOTE 2 The average number of instances of this mechanism in an association is equal to the maximum anticipation (a) divided by the average number of packets necessary for exchange of a service.

Machine: SG_SM

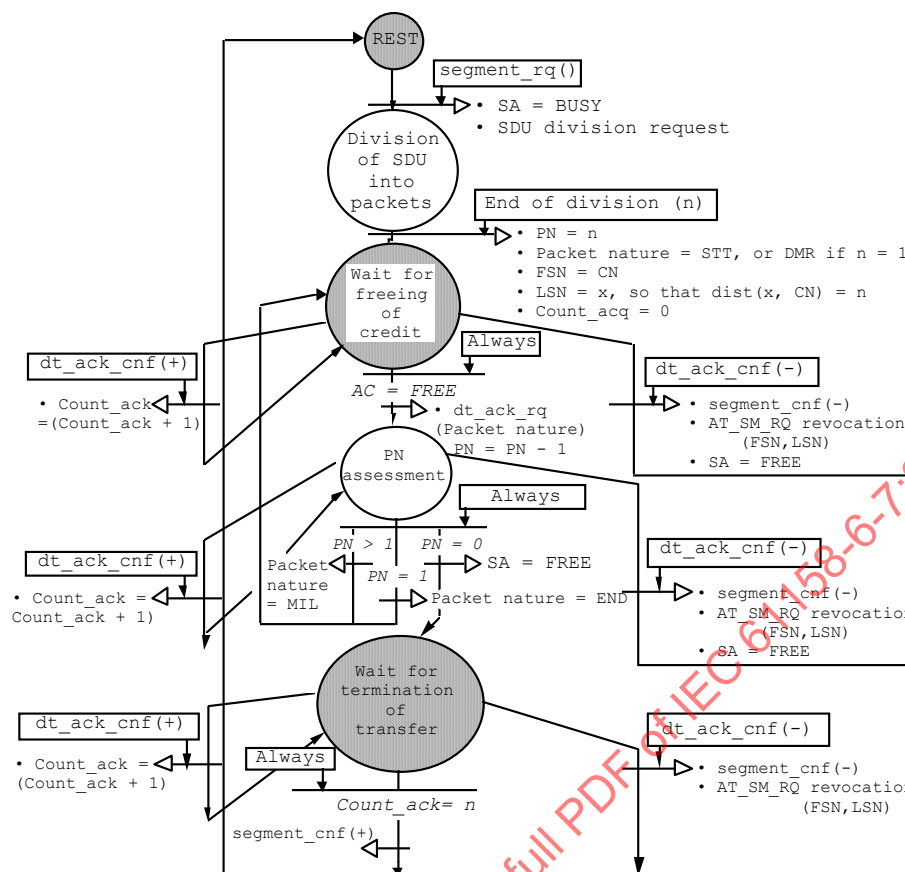


Figure 33 – Segmentation mechanism state machine

NOTE 3 The various mechanisms which can be implemented in an association can, in some cases, operate in parallel. It is therefore necessary to specify that after crossing of a transition, a mechanism retains the new target status until all the actions induced by the transition are terminated. Thus the SG_SM machine retains "PN assessment" status as long as dt_ack_rq(), then number taking, then ma_rq() have not been accounted for by machines ATAK_SM_RQ, NB_SM and MA_SM respectively.

d) Description of mechanism:

As soon as a data transfer service segmentation request is made, service access (SA) is inhibited until all the packets are assigned to an acknowledged data transfer machine. Service access is then freed, allowing new segmentation requests to be accounted for, even if transfer of the packets of the preceding request is not terminated.

Segmentation request processing always begins with division of the MCS-SDU into n packets, starting at the beginning. The different packets are the subject of individual acknowledged transfer. The first packet resulting from division is attributed packet nature STT (or ALL if the MCS-SDU is contained in a single packet) and an FSN sequence number. The following packets have a packet nature of MID, except for the last which has a packet nature of END.

During transmission of the different packets associated with the segmentation of the service, the segmentation machine passes the packets one by one as the acknowledged transfer machine invocations are freed. The availability of acknowledged data transfer machines can be monitored using anticipation credit value FREE.

In addition, if a transmission failure is detected by an acknowledged transfer machine, sending of the remaining packets to be sent is suspended and revocation of the acknowledged transfer machine is effected for each packet in the process of being transferred. This suspension makes it possible to immediately free transfer machines which become available for other SDU exchanges. The acknowledged transfer positive confirmation primitives are counted to detect determination of the transfer of all the packets, and in order to obtain positive confirmation of the segmentation request.

10.2.5.10 Reassembly (RS_SM)

A reassembly element can be implemented in an association. This element makes it possible to reconstitute the data associated with a data transfer service which has been received in a number of packets. Implementation of this element at data transfer receiver level can only be carried out conjointly with that of segmentation at initiator level.

This reassembly element is both reactive and transformational, it is consequently necessary to provide a state machine and describe the associated processing. See Figure 34.

a) Description of variables specific to the mechanism:

The data specific to this mechanism can be represented by a reassembly list as shown in Table 26. This reassembly list is an LR[i] structure table, in which the number of elements is equal to (a+s), where “a” is the association anticipation factor and “s” is the maximum number of MCS-PDUs for an MCS-SDU, in the context of an association. All structures of the table include a sequence number, the packet reassembly status associated with the sequence number as well as the packet nature LR[i] = (sn[i], Etrs[i] and NatP[i]).

Reassembly status can be either:

- UNAVAILABLE, if the packet has not been received and therefore not restored to the user,
- AVAILABLE, if the packet has been received.

The packet nature is either:

- STT, first packet of SDU,
- MID, for intermediate packets of the SDU,
- END, for the last packet of the SDU,

At any given time, the sequence number of the first element of the table is the largest of all those received and has a reassembly status value which is necessarily other than available.

Each of the other elements in the table has a sequence number which decreases by one modulo increment relative to that preceding it, and one of the two possible reassembly status values.

- ALL, for a packet containing the entire SDU.

NOTE The packet nature cannot have the value NIL in the reassembly machine case.

Table 26 – Structure of the reassembly list

Sequence number	Reassembly status	Packet nature
sn(1)	AVAILABLE	STT/MID/END/ALL
sn(2) such that dist(sn(1), sn(2)) = 1	AVAIL/UNAVAIL	STT/MID/END/ALL
.....		
sn(a) such that dist(sn(1), sn(a)) = a-1	AVAIL/UNAVAIL	STT/MID/END/ALL
sn(a+1) such that dist(sn(1), sn(a+1)) = a	AVAIL/UNAVAIL	STT/MID/END/ALL
.....		
sn(a+s) such that dist(sn(1), sn(a+s)) = a+s-1	AVAIL/UNAVAIL	STT/MID/END/ALL

b) The following events and conditions are accounted for by the mechanism:

- dt_ack_ind(Packet nature, NS).
- “End of update”, “End of shift”, “End of operation”, corresponding to termination of the operations.

- Packet nature $\neq$ NIL (and) SN do not belong to “Reassembly list”, so as to be able to ascertain that the packet does in fact originate from a segmentation machine and that the sequence number belongs to the antiduplication window.
- Packet nature $\neq$ NIL (and) SN do not belong to “Reassembly list”, so as to consider a sequence number greater than those received to date.
- Packet nature = END (or) ALL, possibility of detecting an entire message.
- Packet nature = STT (or) MID, impossible to detect an entire message.
- SDU = COMPLETE or INCOMPLETE, reception of all or not all of the packets of a transfer.

c) The following actions are triggered by the mechanism:

- Updating of “reassembly list”, “reassembly list” shift, “reassembly list” scanning, initiation of associated processing,
- reassemb_ind.

d) Mechanism:

For an association in which it is possible to implement reassembly, a single instance of this machine is necessary.

Machine: RS_SM

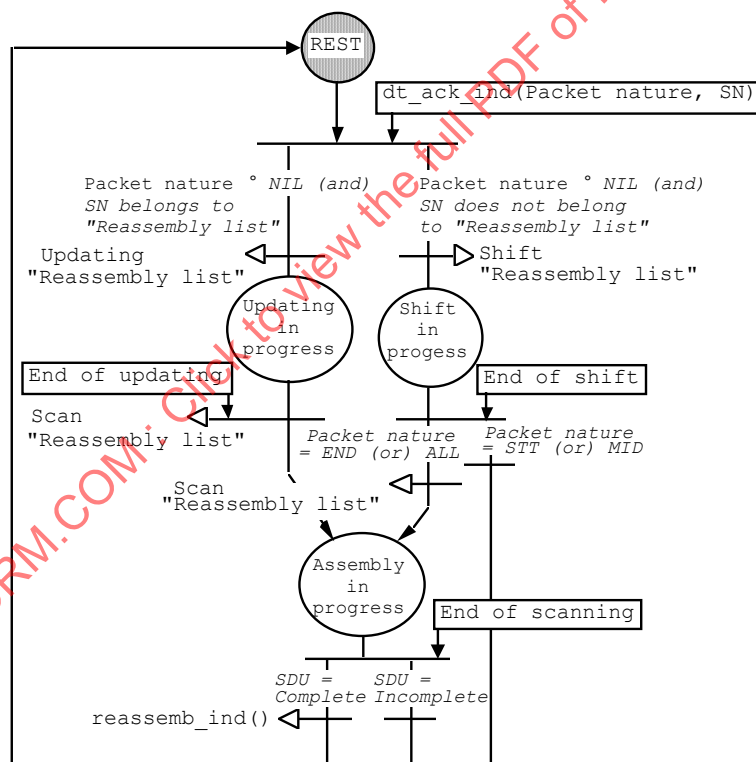


Figure 34 – Reassembly mechanism state machine

e) Description of mechanism:

When a packet is received, depending on whether or not its sequence number belongs to the reassembly list, “Update” or “Shift” processing is carried out on the reassembly list.

In the case where the assembly list shall be shifted whereas the packet nature is STT or MID, no further action is required. In all other cases, the list is scanned for complete SDUs, any found being restored to the user by a reassembly indication.

Description of the processing used by the machine:

- “Update”: Updating of the reassembly list consists of updating the “reassembly status” and “packet nature” fields of that element with the same sequence number as that received.

This updating consists of setting the packet nature to STT, MID, END or ALL depending on the value received, and reassembly status to AVAILABLE.

NOTE 1 The reassembly machine cannot receive the same packet twice as prior filtering is carried out by the antiduplication machine.

- “Shift”: This processing consists of shifting the elements in the reassembly list so that the value of $sn(1)$ corresponds to the SN sequence number received.

This processing involves the following operations, after evaluation of distance $N = dist(SN, sn(1))$.

- if $N < a+s$, then for all the elements of all the $LR[i]$ list, where i decreases from $a+s$ to $N+1$, set $LR[i] = LR[i-N]$. This shifts the elements of the structure by N cells.
- where i increases from 2 to $Min(N, a+s)$ set $sn[i] = M$ so that $dist(SN, M) = i-1$ and $Etrs[i] = UNAVAILABLE$. This is a question of setting to UNAVAILABLE the reception status associated with the sequence numbers contained exclusively within the values of $sn(1)$ and SN .
- set $sn(1) = SN$, and $Etrs(1) = AVAILABLE$, and $NatP(1) = Packet\ nature$. It is a question of setting the first element of the list to AVAILABLE while retaining its packet nature.
- “Scan”: Scanning consists of determining whether the packet received contains an ALL packet nature or scanning the reassembly list on either side of the packet received to find any series of consecutive numbers, marked available, bounded by two packets of packet natures END and STT respectively.
- This processing involves the following operations, after having identified m , such that $sn(m) = SN$:
- if $NatP(m) = ALL$, reconstitute the SDU.
- if $NatP(m) \neq ALL$, set $j = m$ and $Finexp = FALSE$, then perform the following operations until $Finexp \neq FALSE$ or $j = 0$:
- if $Etrs(j) = UNAVAILABLE$, set $Finexp = TRUE$,
- if $Etrs(j) = AVAILABLE$ and $NatP(j) = END$, then set $Finexp = CONTINUE$,
- $j = j + 1$.
- if $Finexp = CONTINUE$, set $k = m$, then perform the following operations until $Finexp \neq CONTINUE$ or $k = a+s$:
- $k = k+1$,
- if $Etrs(k) = UNAVAILABLE$, set $Finexp = TRUE$,
- if $Etrs(k) = AVAILABLE$ and $NatP(k) = STT$, then set $Finexp = TRUE$, reconstitute the SDU.

NOTE 2 The numbers of the first and last packet of the SDU are k and $j+1$ respectively.

10.2.6 Non-associated mode data transfer

10.2.6.1 General

Two protocol procedures can be implemented in the context of non-associated mode data transfer, depending on whether or not acknowledgement is requested. See Figure 35.

NOTE Non-associated mode data transfer mechanisms do not feature retry and antiduplication or segmentation and reassembly.

The following state machines are defined:

- NT_SM_RQ, NT_SM_ACC:
Non-associated mode unacknowledged transfer machine for requester and acceptor.
- NTAK_SM_RQ, NTAK_SM_ACC:
Non-associated mode acknowledged transfer machine for requester and acceptor.

In the non-associated mode, the state machines concerned with these transfers interact with both:

- the other application service elements available to the application entity. These are carried out by local identification,
- and the data link layer in the case where an ADAE type link address pair localises the pair of application entities communicating.

The following diagram shows all the state machines implemented by the non-associated transfers at the level of an application entity, and situates them relative to associated transfers of the same entity.

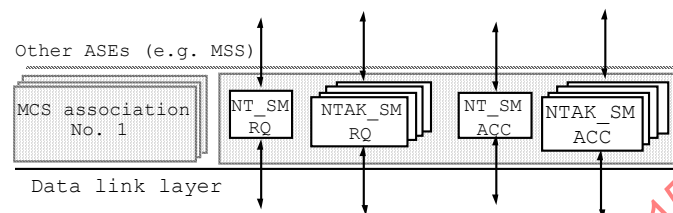


Figure 35 – Interaction of state machine in a non associated data transfer

This shows that an application entity can have a number of associations possessing their own machines, as well as a set of machines delegated to non-associated transfers which are shared by all the corresponding entities.

10.2.6.2 Unacknowledged transfer (NT_SM)

10.2.6.2.1 General

The unacknowledged non-associated mode data transfer procedure consists of two parts, relating to the requester and the acceptor respectively.

10.2.6.2.2 Requester element (NT_SM_RQ)

The requester element is that which causes the sending of a data transfer request from an application entity to a destination in another application entity. See Figure 36.

a) The following events and conditions are accounted for:

- A_UNIDATA.rq (Request_ack), Request_ack = FALSE.
- L_MSG_XX.cnf(+/-) (See IEC 61158-3-7, Type-7)

b) The following actions are performed:

- L_MSG_XX.rq(NTRQ[noack]) (See IEC 61158-3-7, Type-7)
- A_UNIDATA.cnf(+/-).

c) Mechanism:

An application entity in which it is possible to implement the unacknowledged data transfer procedure supports one or possibly a number of instances of the mechanism for local anticipation purposes at device level.

Machine: NT_SM_RQ

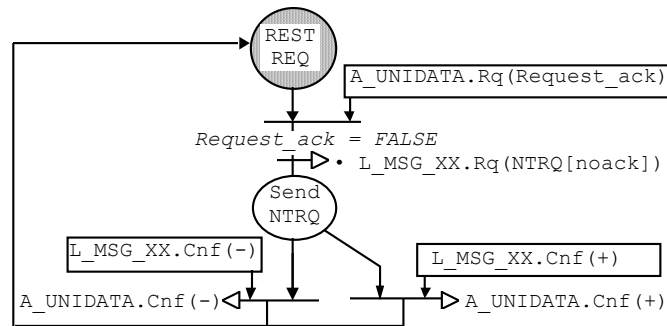


Figure 36 – Unacknowledged transfer: Requester element state machine

d) Description of mechanism:

As soon as an A_UNIDATA.rq(Request_ack) unacknowledged data transfer request is made where (Request_ack = FALSE) with a target application entity, a data transfer (NTRQ) is sent.

Whatever the results of the send, the mechanism returns to its initial state.

10.2.6.2.3 Acceptor element (NT_SM_ACC)

The acceptor element is that which receives a data transfer for which acknowledgement is not requested and transfers it to the user. See Figure 37.

a) The following events and conditions are accounted for:

- L_MSG_XX.ind(NTRQ[noack]) (See IEC 61158-3-7, Type-7)

b) The following actions are carried out by the acceptor element:

- A_UNIDATA.ind (Request_ack = FALSE)

c) Acceptor element mechanism:

An application entity in which it is possible to implement the unacknowledged data transfer procedure supports one or possibly a number of instances of the mechanism for local anticipation purposes.

Machine: NT_SM_ACC

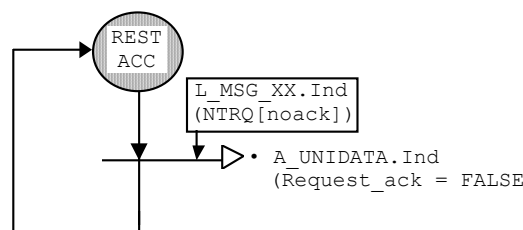


Figure 37 – Unacknowledged transfer: Acceptor element state machine

d) Description of mechanism:

On receiving an L_MSG_XX.ind() primitive whose data correspond to an NTRQ for which acknowledgement is not requested, when the application entity has the resources necessary for instantiation of the mechanism, the primitive A_UNIDATA.ind(Request_ack = FALSE) is invoked.

NOTE The resources associated with a mechanism are, indeed, shared between all the correspondents of the application entity.

10.2.6.3 Acknowledged transfer (NTAK_SM)

10.2.6.3.1 General

The acknowledged non-associated mode data transfer procedure consists of two parts, relating respectively to the requester and the acceptor.

10.2.6.3.2 Requester element (NTAK_SM_RQ)

The requester element is that which causes the sending of a data transfer request from one application entity to a destination in another application entity and which receives the corresponding acknowledgement. See Figure 38.

a) The following events and conditions are accounted for:

- A_UNIDATA.rq (Request_ack = TRUE),
- L_MSG_XX.cnf(+/-) (See IEC 61158-3-7, Type-7),
- reception of an AKNT in the data of the L_MSG_XX.ind primitive containing an acknowledgement request,
- acknowledgement time-out.

b) The following actions are carried out by the requester element:

- calling of the "Identification" function to assign an invocation identification to each transfer,
- L_MSG_XX.rq(NTRQ[ack]) (See IEC 61158-3-7, Type-7),
- starting of acknowledgement time-out,
- A_UNIDATA.cnf(+/-).

c) Requester element mechanism:

For an application entity in which it is possible to implement the acknowledged data transfer procedure, a number of instances of the mechanism can exist simultaneously with one or more per destination application entity.

Machine: NTAK_SM_RQ

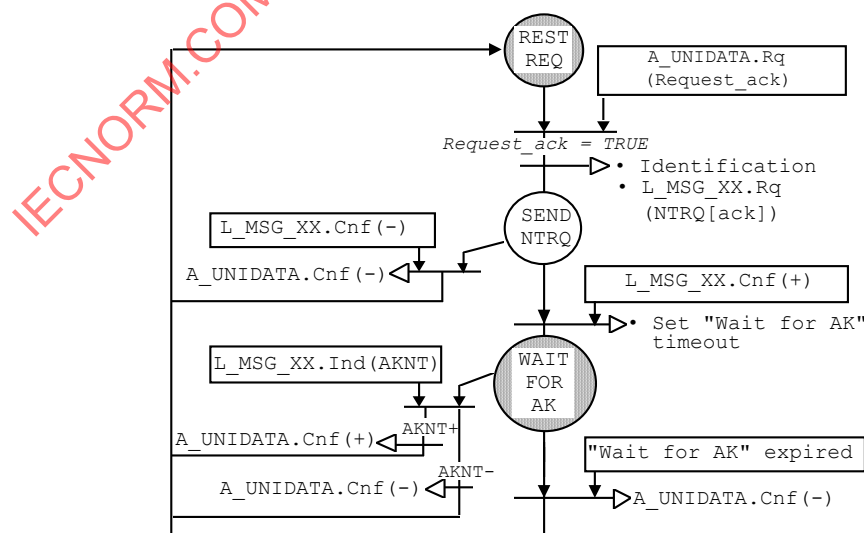


Figure 38 – Acknowledged transfer: Requester element state machine

d) Description of mechanism:

As soon as an acknowledged data transfer is requested, an NTRQ is sent.

When the result of the send returned by the data link layer is positive, an acknowledgement time-out is started. Either the expected acknowledgement arises and the mechanism returns to its initial state after returning a confirmation to the user which can be positive or negative depending on whether or not the acknowledgement received is positive, or the acknowledgement time-out expires and a negative confirmation is sent to the user.

10.2.6.3.3 Acceptor element (NTAK_SM_ACC)

The acceptor element is that which receives the data transfer and which sends the corresponding acknowledgement, and relays the data to the user. See Figure 39.

a) The following events and conditions are accounted for:

- L_MSG_XX.ind(NTRQ[ack]) (See IEC 61158-3-7, Type-7)
- L_MSG_XX.cnf(+/-) (See IEC 61158-3-7, Type-7)

b) The following actions are carried out by the requester element:

- L_MSG_XX.rq(AKNT[+/-]) (See IEC 61158-3-7, Type-7)
- A_UNIDATA.ind(Request_ack = TRUE)

c) Acceptor element mechanism:

For an entity in which it is possible to implement the acknowledged data transfer procedure, a number of instances of the machine can exist simultaneously with one or more per source entity.

Machine: NTAK_SM_ACC

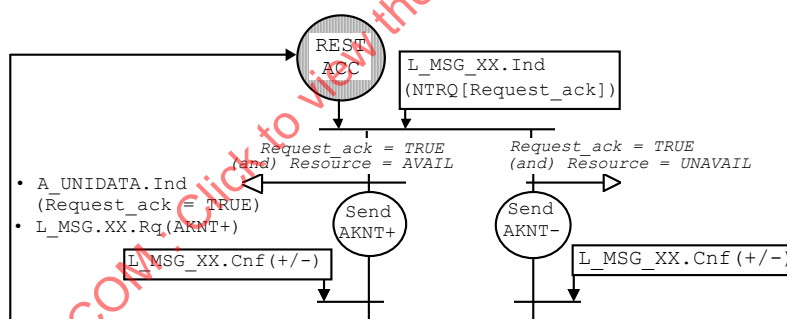


Figure 39 – Acknowledged transfer: Acceptor element state machine

d) Description of mechanism:

As soon as the L_MSG_XX.ind primitive, the data of which corresponds to an NTRQ with an acknowledgement request, the data transfer sender is returned an acknowledgement containing a sequence number equal to that of the acknowledged data transfer.

Depending on whether or not the message can be supplied to the user, the acknowledgement return is positive or negative. The various reasons preventing an application entity from returning data to the user are modelled by the resources not being available.

Finally, whether or not there is a failure during sending of the acknowledgement at link level, the mechanism returns to its initial state.

10.2.6.4 Invocation identification

When using the acknowledged data transfer service with a pair of application entities, the identification function supplies an invocation identification which transits with the data in the transfers using the NTRQ type PDU, and is also found in the corresponding acknowledgements of which the PDU is of the AKNT type.

There is a single instance of this mechanism per application entity. The value of the invocation identification is random, provided it makes it possible to discriminate between the different transfers pending acknowledgement in the same pair of application entities.

NOTE There is no special initial value or rules for changing the value of the invocation identification.

10.2.6.5 Anticipation

Multiple anticipation is authorised in the case of acknowledged data transfers, provided the resources are available at the requester and the acceptor.

11 Protocol options

11.1 Conformance classes

11.1.1 General

Conformance classes in AL define various levels of quality of service. These levels are qualified in terms of the services provided by a device and in terms of the supported mechanisms. Three series of parameters are used to define the conformance class. These characterize the horizontal, vertical and type conformance.

11.1.2 Vertical conformance

11.1.2.1 General

Vertical conformance describes the application level services supported by a device.

To define the device vertical conformance there are three independent parameters on which some constraints apply. A set of values of those parameters defines a vertical conformance block. Each parameter is representative of a functionality or group of functionalities provided by the application level services, and it has a set of values featuring the ability of the device to support these functionalities.

The values of those parameters are set in upward compatibility. So, a device supporting the services characterized by the parameter value *n* will also support all the functionalities provided by services characterized by values inferior to *n* of this same parameter.

For the AL TYPE 7 application layer, the parameters are:

11.1.2.2 Read / write parameter: PV_R/W

This parameter defines the various read / write possibilities supported by the device depending on the provided services. See Table 27. It can take the following values.

PV_R/W[1] Local write Service (Primitive A_WRITELOC)

PV_R/W[2] is equal to PV_R/W[1] plus local read Service (Primitive A_READLOC)

PV_R/W[3] is equal to PV_R/W[2] plus update Service (Primitive A_UPDATE)

PV_R/W[4] is equal to PV_R/W[3] plus remote read and write Services (Primitives A_READFAR plus A_WRITEFAR)

PV_R/W[5] is equal to PV_R/W[4] plus universal read and write Services (Primitives A_READ and A_WRITE).

Table 27 – PV_R/W parameter values

Parameter PV_R/W	1	2	3	4	5
A_WRITELOC	X	X	X	X	X
A_READLOC		X	X	X	X
A_UPDATE			X	X	X
A_WRITEFAR				X	X
A_READFAR				X	X
A_WRITE					X
A_READ					X

11.1.2.3 Indication parameter: PV_IND

This parameter defines the various possibilities of the indication services supported by the application layer. See Table 28. It can take the following values:

PV_IND[0] No indication service supported

PV_IND[1] Reception and transmission indication services (Primitives A_SENT and A_RECEIVED)

Table 28 – PV_IND parameter values

Parameter PV_IND	0	1
A_SENT		X
A_RECEIVED		X

11.1.2.4 List service parameter: PV_LIS

This parameter defines the access services to variable lists supported by the application layer. See Table 29.

PV_LIS[0] No list access services.

PV_LIS[1] Read variable list service (Primitive A_READLIST)

Table 29 – PV_LIS parameter values

Parameter PV_LIS	0	1
A_READLIST		X

11.1.2.5 Constraints on parameters

11.1.2.5.1 General

For each parameter, a table specifies the constraints by giving the minimum value of the other parameters to which this parameter is linked.

11.1.2.5.2 Parameter PV_R/W

No constraint.

11.1.2.5.3 Parameter PV_IND

No constraint.

11.1.2.5.4 Parameter PV_LIS

See Table 30 for the definitions.

Table 30 – Constraints on PV_LIS parameter

Constraints on PV_LIS	0	1
PV_R/W		PV_R/W[2]

11.1.2.5.5 Vertical Conformance parameters for Sub-MMS ASEs

The vertical parameters make it possible to specify the subset of primitives of services offered.

a) See Table 31 for the definitions of the associated mode data transfer parameter.

Table 31 – PV_AT parameter values

PV_AT	1	2	3	
A_ASSOCIATE			X	
A_ABORT			X	
A_DATA		X	X	

b) See Table 32 for the definitions of the association termination parameter.

Table 32 – PV_RE parameter values

PV_RE	1	2		
A_RELEASE		X		

c) See Table 33 for the definitions of the non-association mode data transfer parameter.

Table 33 – PV_UT parameter values

PV_UT	1	2		
A_UNIDATA		X		

See Table 34 for the expression of the constraints which remain between the various vertical parameters.

Table 34 – Constraints on PV_RE parameter

Constraints on PV_RE	1	2		
PV_AT		[3]		

11.1.3 Horizontal conformance

11.1.3.1 Asynchronous refreshment parameter: PH_R_A

This parameter, as shown in Table 35, describes the asynchronous refreshment status elaboration mechanisms that are supported by an information producer. It can take the following values:

PH_R_A[0] No asynchronous refreshment status is elaborated.

PH_R_A[1] A mechanism to elaborate the asynchronous refreshment status is implemented.

Table 35 – PH_R_A parameter values

Parameter PH_R_A	0	1
Asynchronous refreshment		X

11.1.3.2 Synchronous refreshment parameter: PH_R_S

This parameter, as shown in Table 36, describes the synchronous refreshment status elaboration mechanisms that are supported by an information producer. It can take the following values:

PH_R_S[0] No synchronous refreshment status is elaborated.

PH_R_S[1] A mechanism to elaborate the synchronous refreshment status is implemented.

Table 36 – PH_R_S parameter values

Parameter PH_R_S	0	1
Synchronous refreshment		X

11.1.3.3 Punctual refreshment parameter: PH_R_P

This parameter, as shown in Table 37, describes the punctual refreshment status elaboration mechanisms, supported by the application layer of an information producer. It can take the following values:

PH_R_P[0] No punctual refreshment status is elaborated.

PH_R_P[1] A mechanism to elaborate the punctual refreshment status is implemented.

Table 37 – PH_R_P parameter values

Parameter PH_R_P	0	1
Punctual refreshment		X

11.1.3.4 Asynchronous promptness parameter: PH_P_A

This parameter, as shown in Table 38, describes the asynchronous promptness status elaboration mechanisms that are supported by an information producer. It can take the following values:

PH_P_A[0] No asynchronous promptness status is elaborated.

PH_P_A[1] A mechanism to elaborate the asynchronous promptness status is implemented.

Table 38 – PH_P_A parameter values

Parameter PH_P_A	0	1
Asynchronous promptness		X

11.1.3.5 Synchronous promptness parameter: PH_P_S

This parameter, as shown in Table 39, describes the synchronous promptness status elaboration mechanisms that are supported by an information producer. It can take the following values:

PH_P_S[0] No synchronous promptness status is elaborated.

PH_P_S[1] A mechanism to elaborate the synchronous promptness status is implemented.

Table 39 – PH_P_S parameter values

Parameter PH_P_S	0	1
Synchronous promptness		X

11.1.3.6 Punctual promptness parameter: PH_P_P

This parameter, as shown in Table 40, describes the variable punctual promptness status elaboration mechanisms supported by the application layer of an information consumer. It can take the following values:

PH_P_P[0] No punctual promptness status is elaborated.

PH_P_P[1] A mechanism to elaborate the punctual promptness status is implemented.

Table 40 – PH_P_P parameter values

Parameter PH_P_P	0	1
Punctual promptness		X

11.1.3.7 Variables list consistency parameter: PH_COH

This parameter, as shown in Table 41, describes the various production or transmission status elaboration mechanisms associated with a variable list at the level of a consumer entity. It can take the following values:

PH_COH[0] No consistency mechanism is supported

PH_COH[1] Consistency status elaboration mechanisms implemented

Table 41 – PH_COH parameter values

Parameter PH_COH	0	1
Consistency		X

11.1.3.8 List reliability parameter: PH_FIA

This parameter, as shown in Table 42, describes the various mechanisms supported to make the variable transmission reliable at the consuming application entity level.

PH_FIA[0] No reliability mechanism is supported

PH_FIA[1] List recovery mechanism implemented

Table 42 – PH_FIA parameter values

Parameter PH_FIA	0	1
Recovery		X

11.1.3.9 Transitory spatial consistency status: PH_SPF

This parameter, as shown in Table 43, describes the elaboration mechanism of the contents of the produced consistency variable. This variable is used in the elaboration of the transitory spatial consistency status of the lists of variables at the consumer application entity level.

PH_SPF[0] No transitory spatial consistency mechanism is supported

PH_SPF[1] A mechanism of transitory spatial consistency on lists is implemented

Table 43 – PH_SPF parameter values

Parameter PH_SPF	0	1
Transitory spatial consist.		X

11.1.3.10 Permanent spatial consistency parameter: PH_SPM

This parameter, as shown in Table 44, describes the contents elaboration of the produced consistency variable mechanism. This variable is used in the elaboration of the permanent spatial consistency status of the lists of variables at the consumer application entity level.

PH_SPM[0] No permanent spatial consistency mechanism is supported

PH_SPM[1] A mechanism of permanent spatial consistency on lists is implemented

Table 44 – PH_SPM parameter values

Parameter PH_SPM	0	1
Permanent spatial consist.		X

11.1.3.11 Access parameter: PH_ACC

This parameter, as shown in Table 45, describes the various variable access modes. It can take the following values:

PH_ACC[0] Application variable global access

PH_ACC[1] is equal to PH_ACC[0] plus variable partial access

PH_ACC[2] is equal to PH_ACC[1] plus variable renamed access

Table 45 – PH_ACC parameter values

Parameter PH_ACC	0	1	2
Partial access		X	X
Renamed access			X

11.1.3.12 Resynchronization parameter: PH_RES

This parameter, as shown in Table 46, defines whether or not the resynchronization mechanism is supported by the application layer. It can take the following values:

PH_RES[0] The resynchronization mechanism is not supported

PH_RES[1] The resynchronization mechanism is supported

Table 46 – PH_RES parameter values

Parameter PH_RES	0	1
Resynchronization		X

11.1.3.13 Horizontal conformance parameters for Sub-MMS AEs

The horizontal parameters make it possible to specify the various procedures supported.

a) Acknowledgement parameter: See Table 47.

Table 47 – PH_AK parameter values

PH_AK	1	2		
AT_SM		C		
NT_SM		C		
NB_SM		C		
MA_SM		C		

As soon as PV_AT and PV_UT have value 2, if PH_AK has the value 2, they shall offer the acknowledged transfer.

b) Retry/antiduplication parameter: See Table 48.

Table 48 – PH_RA parameter values

PH_RA	1	2		
RC_SM		X		
DU_SM		X		

c) Segmentation/reassembly parameter: See Table 49.

Table 49 – PH_SR parameter values

PH_SR	1	2		
SG_SM		X		
RS_SM		X		

d) Flow control parameter: See Table 50.

Table 50 – PH_CF parameter values

PH_CF	1	2		
Flow control		X		

Expression of the constraints which remain between the various horizontal parameters. See Table 51 and Table 52.

Table 51 – Constraints on PH_RA parameter

Constraints on PH_RA	1	2		
PH_AK		[2]		

Table 52 – Constraints on PH_SR parameter

Constraints on PH_SR	1	2		
PH_AK		[2]		

11.1.4 Type conformance

11.1.4.1 General

The set of types available for the AL TYPE application variables is issued from all the possible instances of the Type Constructor object. Within this set are defined a certain number of subsets related to the various categories of variables handled by a device. The conformance type blocks define the supported types subsets on a device.

11.1.4.2 Primitive type parameters

11.1.4.2.1 Octet string parameter PT_OCT

This parameter specifies whether or not the device supports the octet string type. See Table 53.

PT_OCT[0] Octet string type not supported

PT_OCT[1] Octet string type supported

Table 53 – PT_OCT parameter values

Parameter PT_OCT	0	1
Octet string		X

11.1.4.2.2 Binary string parameter PT_BIN

This parameter specifies whether or not the device supports the binary string type. See Table 54.

PT_BIN[0] Binary string type not supported

PT_BIN[1] Binary string type supported

Table 54 – PT_BIN parameter values

Parameter PT_BIN	0	1
Bit string		X

11.1.4.2.3 Visible string parameter PT_VIS

This parameter specifies whether or not the device supports the visible string type. See Table 55.

PT_VIS[0] Visible string type not supported

PT_VIS[1] Visible string type supported

Table 55 – PT_VIS parameter values

Parameter PT_VIS	0	1
Visible string		X

11.1.4.2.4 Boolean parameter PT_BOO

This parameter specifies whether or not the device supports the boolean type. See Table 56.

PT_BOO[0] Boolean type not supported

PT_BOO[1] Boolean type supported

Table 56 – PT_BOO parameter values

Parameter PT_BOO	0	1
Boolean		X

11.1.4.2.5 BCD parameter PT_BCD

This parameter specifies whether or not the device supports the BCD type. See Table 57.

PT_BCD[0] BCD type not supported

PT_BCD[1] BCD type supported

Table 57 – PT_BCD parameter values

Parameter PT_BCD	0	1
BCD		X

11.1.4.2.6 Binary time parameter PT_BTM

This parameter specifies whether or not the device supports the binary time type. See Table 58.

PT_BTM[0] Binary time type not supported

PT_BTM[1] Binary time type supported

Table 58 – PT_BTM parameter values

Parameter PT_BTM	0	1
Binary time		X

11.1.4.2.7 Integer parameter PT_INT

This parameter specifies whether or not the device supports the integer type. See Table 59.

PT_INT[0] Integer type not supported

PT_INT[1] Integer type supported

Table 59 – PT_INT parameter values

Parameter PT_INT	0	1
Integer		X

11.1.4.2.8 Unsigned parameter PT_UNUS

This parameter specifies whether or not the device supports the unsigned type. See Table 60.

PT_UNUS[0] Unsigned type not supported

PT_UNUS[1] Unsigned type supported

Table 60 – PT_UNUS parameter values

Parameter PT_UNUS	0	1
Unsigned		X

11.1.4.2.9 Floating point parameter PT_FPT

This parameter specifies whether or not the device supports the floating point type. See Table 61.

PT_FPT[0] Floating point type not supported

PT_FPT[1] Floating point type supported

Table 61 – PT_FPT parameter values

Parameter PT_FPT	0	1
Floating point		X

11.1.4.2.10 Generalized time PT_GTM

This parameter specifies whether or not the device supports the generalized time type. See Table 62.

PT_GTM[0] Generalized time type not supported

PT_GTM[1] Generalized time type supported

Table 62 – PT_GTM parameter values

Parameter PT_GTM	0	1
Generalised time		X

11.1.4.3 Array type parameter PT_TAB

This parameter describes the various possibilities for a device to implement the array type. See Table 63. It can take the following values:

PT_TAB[0] Array type not supported

PT_TAB[1] Array composed of primitive types

PT_TAB[2] is equal to PT_TAB[1] plus complex structured type arrays (array, structures)

Table 63 – PT_TAB parameter values

Parameter PT_TAB	0	1	2
Primitive type array		X	X
Structured type array			X

11.1.4.4 Structure type parameter PT_STR

This parameter describes the several possibilities for a device to implement the structure type. See Table 64. It can take the following values:

PT_STR[0] Structure type not supported

PT_STR[1] Structure composed of primitive types

PT_STR[2] is equal to PT_STR[1] plus complex structured type structures (array, structures)

Table 64 – PT_STR parameter values

Parameter PT_STR	0	1	2
Primitive type structure		X	X
Complex type structure			X

11.1.4.5 Constraints on parameters**11.1.4.5.1 General**

For each parameter, a table specifies the constraints by giving the minimum value of the other parameters to which it is linked.

11.1.4.5.2 Parameter PT_TAB

See Table 65.

Table 65 – Constraints on PT_TAB parameter

Constraints on PT_TAB	0	1	2
PT_TAB			PT_TAB[1]

11.1.4.5.3 Parameter PT_STR

See Table 66.

Table 66 – Constraints on PT_STR parameter

Constraints on PT_STR	0	1	2
PT_STR			PT_STR[1]

There are no particular constraints on the other parameters specifying type conformity.

11.1.5 Protocol Implementation Conformance statement

11.1.5.1 General

This part of the document describes the "Protocol Implementation Conformance Statement Proforma (PICS). Each implementation should fill all the PICS.

- 1- Part 1 of the PICS shall indicate the information relative to the implementation and to the system.
- 2- Part 2 of the PICS shall indicate the CBB services implemented
- 3- Part 3 of the PICS shall indicate the CBB services and the values implemented.
- 4- Part 4 of the PICS shall indicate the "local implementations values".

11.1.5.2 PICS Part 1 :

PICS Serial Number :

Date:

Vendor's Name,

Model Name,

Revision Identifier,

Machine Name(s) and Version Number(s),

Operating System(s),

Sub-mms Abstract syntax,

Version Number of the Sub-MMS Abstract Syntax Supported (minor number),

CS Sub-mms abstract syntax,

Version Number of the CS Sub-MMS Abstract Syntax Supported (minor number),

Calling (yes or no),

Called (yes or no),

List of standardized names.

11.1.5.3 PICS part 2 :

PICS Serial Number :

Date issued :

Initiate,

(server,client or both)

Conclude,

Unsolicited Status,

Identify

Status,

Get name list,

Delete domain,

Initiate download sequence,

Download segment,

Terminate download sequence,

Initiate upload sequence,

Upload segment,

Terminate upload sequence,

Get domain attributes,

Create program invocation,

Delete program invocation,

Start,

Stop,

Resume,

Reset,

Kill,

Get program invocation attributes,

Read,

Write,

Get variable access attributes,

Information report,

Define variable list,

Delete variable list,

Get variable list attributes,

Event notification,

Get alarm summary,

Alter event condition monitoring,

Acknowledge event notification,

Get event condition attributes.

Generic Init Download

Generic Download

Generic Terminate Download

Get OD Header/Data Type Request

11.1.5.4 PICS part 3 :

PICS Serial number :

Date :

Object identification by name

Object identification by index

Recursivity of variable data

variable-list

Partial access

11.1.5.5 PICS part 4 :

PICS Serial number :

Date :

List of Data Types supported with limit values

Max time-out value in seconds,

Time with system-clock,

Granularity of time in milliseconds,

Uninterruptible-access to variables,

Local detail & execution argument for all services,

Max outstanding calling,

Max outstanding called,

Load data format for Download segment,

Max number of simultaneous upload.

Max number of variables read in one service invocation (1 to n)

Max number of variables written in one service invocation (1 to n)

Max number of variables notified in one service invocation (1 to n)

Max number of events notified in one service invocation (0 to n)

Max number of events read in one service invocation (1 to n)

Maximum number of acknowledged event objects (1 to n)

Maximum number of modified event objects (1 to n)

Object Access Protection (supported or not)

11.1.5.6 Minimum conformance classes

The following conformance classes concern only the SERVER devices.

The minimum conformance classes are listed by management types.

The types of management are:

- a) association management,
- b) VMD management,
- c) domain management,
- d) program invocation management,
- e) variable management,
- f) event management.

There are 5 conformance classes for each type of management. Each of these conformance classes defines the services, the data types, the complex variable objects and the obligatorily supported mechanisms.

The conformance class of a device is composed of 6 values. Each value corresponds to the conformance class of one type of management. These 6 values can be different from one another.

It is not forbidden to have a device belonging to one conformance class services, data types, complex variable objects and mechanisms not obligatory for its class.

1- Conformance classes for environment management: See Table 67.

Table 67 – Conformance classes for environment management

Services	Class 1	Class 2	Class 3	Class 4	Class 5
initiate				M	M
conclude					M
abort				M	M
status					
unsolicited status					
identify					
get name list					
initiate download sequence					
download segment					
terminate download sequence					
initiate upload sequence					
upload segment					
terminate upload sequence					
delete domain					
get domain attributes					
create program invocation					
delete program invocation					
start					
stop					
resume					
reset					
kill					
get program invocation attributes					
read					
write					
information report					
define variable list					
delete variable list					
get variable access attributes					
get variable list attributes					
event notification					
acknowledge event notification					
alter event condition monitoring					
get alarm summary					
get event condition attributes					
Variable Data Type					
simple data					
structured data (arr/str)					
Complex Variable Objects					
Variable-list					
Mechanisms					
partial access to the variables					
controlled access by protections					
association(predef/negotiated)		M	M	M	M
without association	M		M	M	M

2- Conformance classes for VMD management: See Table 68.

Table 68 – Conformance classes for VMD management

Services	Class 1	Class 2	Class 3	Class 4	Class 5
initiate					
conclude					
abort					
status		M	M	M	M
unsolicited status				M	M
identify			M	M	M
get name list					M
initiate download sequence					
download segment					
terminate download sequence					
initiate upload sequence					
upload segment					
terminate upload sequence					
delete domain					
get domain attributes					
create program invocation					
delete program invocation					
start					
stop					
resume					
reset					
kill					
get program invocation attributes					
read					
write					
information report					
define variable list					
delete variable list					
get variable access attributes					
get variable list attributes					
event notification					
acknowledge event notification					
alter event condition monitoring					
get alarm summary					
get event condition attributes					
Variable Data Type					
simple data					
structured data (arr/str)					
Complex Variable Objects					
Variable-list					
Mechanisms					
partial access to the variables					
controlled access by protections					
association(predef/negotiated)					
without association					

3- Conformance classes for domain management: See Table 69.

Table 69 – Conformance classes for PI management

Services	Class 1	Class 2	Class 3	Class 4	Class 5
initiate					
conclude					
abort					
status					
unsolicited status					
identify					
get name list					
initiate download sequence		M	M	M	M
download segment		M	M	M	M
terminate download sequence		M	M	M	M
initiate upload sequence			M	M	M
upload segment			M	M	M
terminate upload sequence			M	M	M
delete domain				M	M
get domain attributes					M
create program invocation					
delete program invocation					
start					
stop					
resume					
reset					
kill					
get program invocation attributes					
read					
write					
information report					
define variable list					
delete variable list					
get variable access attributes					
get variable list attributes					
event notification					
acknowledge event notification					
alter event condition monitoring					
get alarm summary					
get event condition attributes					
Variable Data Type					
simple data					
structured data (arr/str)					
Complex Variable Objects					
Variable-list					
Mechanisms					
partial access to the variables					
controlled access by protections					M
association(predef/negotiated)					
without association					

4- Conformance classes for PI management: See Table 70.

Table 70 – Conformance classes for domain management

Services	Class 1	Class 2	Class 3	Class 4	Class 5
initiate					
conclude					
abort					
status					
unsolicited status					
identify					
get name list					
initiate download sequence					
download segment					
terminate download sequence					
initiate upload sequence					
upload segment					
terminate upload sequence					
delete domain					
get domain attributes					
create program invocation				M	M
delete program invocation				M	M
start		M	M	M	M
stop		M	M	M	M
resume		M	M	M	M
reset			M	M	M
kill				M	M
get program invocation attributes					M
read					
write					
information report					
define variable list					
delete variable list					
get variable access attributes					
get variable list attributes					
event notification					
acknowledge event notification					
alter event condition monitoring					
get alarm summary					
get event condition attributes					
Variable Data Type					
simple data					
structured data (arr/str)					
Complex Variable Objects					
Variable-list					
Type of access					
partial access to the variables					
controlled access by protections					M
association(predef/negotiated)					
without association					

5- Conformance classes for the variable/variable-list management: See Table 71.

Table 71 – Conformance classes for variable/variable list management

Services	Class 1	Class 2	Class 3	Class 4	Class 5
initiate					
conclude					
abort					
status					
unsolicited status					
identify					
get name list					
initiate download sequence					
download segment					
terminate download sequence					
initiate upload sequence					
upload segment					
terminate upload sequence					
delete domain					
get domain attributes					
create program invocation					
delete program invocation					
start					
stop					
resume					
reset					
kill					
get program invocation attributes					
read	M	M	M	M	M
write	M	M	M	M	M
information report	M	M	M	M	M
define variable list				M	M
delete variable list				M	M
get variable access attributes					M
get variable list attributes					M
event notification					
acknowledge event notification					
alter event condition monitoring					
get alarm summary					
get event condition attributes					
Variable Data Type					
simple data	M	M	M	M	M
structured data (arr/str)		M	M	M	M
Complex Variable Objects					
Variable-list				M	M
Type of access					
partial access to the variables			M	M	M
controlled access by protections					M
association(predef/negotiated)					
without association					

6- Conformance classes for event management: See Table 72.

Table 72 – Conformance classes for event management

Services	Class 1	Class 2	Class 3	Class 4	Class 5
initiate					
conclude					
abort					
status					
unsolicited status					
identify					
get name list					
initiate download sequence					
download segment					
terminate download sequence					
initiate upload sequence					
upload segment					
terminate upload sequence					
delete domain					
get domain attributes					
create program invocation					
delete program invocation					
start					
stop					
resume					
reset					
kill					
get program invocation attributes					
read					
write					
information report					
define variable list					
delete variable list					
get variable access attributes					
get variable list attributes					
event notification		M	M	M	M
acknowledge event notification			M	M	M
alter event condition monitoring				M	M
get alarm summary					M
get event condition attributes					M
Variable Data Type					
simple data					
structured data (arr/str)					
Complex Variable Objects					
Variable-list					
Mechanisms					
partial access to the variables					
controlled access by protections					M
association(predif/negotiated)					
without association					

Bibliography

IEC/TR 61158-1 (Ed.2.0), *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61784-1 (Ed.2.0), *Industrial communication networks – Profiles – Part 1: Fieldbus profiles*

ISO/IEC 7498-3, *Information technology – Open Systems Interconnection – Basic Reference Model — Part 3: Naming and addressing*

ISO 8649, *Information technology – Open Systems Interconnection – Service definition for the Association Control Service Element*

ISO/IEC 8650 (all parts), *Information technology – Open Systems Interconnection – Connection-oriented protocol for the Association Control Service Element: Protocol specification*

ISO/IEC 8822, *Information technology – Open Systems Interconnection – Presentation service definition*

IECNORM.COM : Click to view the full PDF of IEC 61158-6-7:2007

SOMMAIRE

AVANT-PROPOS	160
INTRODUCTION.....	162
1 Domaine d'application	163
1.1 Généralités.....	163
1.2 Spécifications.....	163
1.3 Conformité	164
2 Références normatives.....	164
3 Termes, définitions, symboles, abréviations et conventions	164
3.1 Termes et définitions issus des autres normes ISO/CEI	165
3.2 Termes et définitions issus de la CEI 61158-5-7	166
3.3 Termes et définitions supplémentaires	166
3.4 Abréviations et symboles.....	170
3.5 Conventions	170
3.6 Conventions utilisées dans les diagrammes d'états	171
4 Syntaxe abstraite de type de données	172
4.1 Spécification relative à la syntaxe abstraite de données	172
4.2 Syntaxe abstraite de PDU FAL	175
5 Syntaxe de transfert	176
5.1 Codage compact	176
5.2 Codage de type de données	177
6 Structure des machines de protocole.....	239
7 Diagramme d'états d'AP-context.....	240
8 Machine de protocole des services de la FAL Sub-MMS (FSPM)	241
8.1 Généralités.....	241
8.2 Projection des PDU SUB-MMS sur les services de MCS	241
8.3 Projection du service abort de SUB-MMS sur les services de MCS	241
8.4 Construction d'une PDU-SUB-MMS à partir d'une primitive de service	241
8.5 Extraction d'une primitive de service valide à partir d'une PDU SUB-MMS	241
8.6 Négociation d'une syntaxe abstraite et d'une syntaxe de transfert généralement appelée contexte de présentation.....	242
8.7 Identification de la syntaxe abstraite de SUB-MMS core.....	244
8.8 Identification du nom de contexte d'application.....	244
8.9 Identification de l'ASE de la syntaxe abstraite centrale et de la syntaxe de transfert	244
9 Machine de protocole des relations entre associations (ARPM)	245
10 Machine de protocole de mapping de la DLL (DMPM).....	245
10.1 ARPM et DMPM de MPS	245
10.2 ARPM et DMPM de MCS	258
11 Option de protocole	297
11.1 Classes de conformité	297
Bibliographie.....	318
Figure 1 – Exemple de réseau d'évaluation.....	171
Figure 2 – Codage d'une CompactValue	177

Figure 3 – Organisation des bits et des octets au sein d'une PDU	179
Figure 4 – Codage de type bitstring	182
Figure 5 – Codage d'une valeur de type floating point.....	184
Figure 6 – Codage de type structure	185
Figure 7 – Codage de type Boolean array	186
Figure 8 – Représentation d'une PDU MCS	192
Figure 9 – Relations entre les Machines de Protocole et les Couches Adjacentes.....	239
Figure 10 – Réseau d'évaluation du service A_Readloc.....	246
Figure 11 – Réseau d'évaluation du service A_Writeloc.....	247
Figure 12 – Réseau d'évaluation du service A_Update	248
Figure 13 – Réseau d'évaluation du service A_Readfar	250
Figure 14 – Réseau d'évaluation du service A_Writefar	252
Figure 15 – Réseau d'évaluation du service A_Sent	253
Figure 16 – Réseau d'évaluation du service A_Received	253
Figure 17 – Établissement d'association: diagramme d'états de l'élément demandeur	260
Figure 18 – Établissement d'association: diagramme d'états de l'élément répondeur	262
Figure 19 – Interruption d'association: diagramme d'états de l'élément demandeur	264
Figure 20 – Interruption d'Association: diagramme d'états de l'élément répondeur.....	266
Figure 21 – Révocation d'association: diagramme d'états de l'élément demandeur.....	268
Figure 22 – Révocation d'association: diagramme d'états de l'élément accepteur	269
Figure 23 – Interactions entre les diagrammes d'états dans un transfert de données en mode associé	270
Figure 24 – Service de transfert: diagramme d'états de l'élément demandeur	275
Figure 25 – Service de transfert: diagramme d'états de l'élément accepteur	276
Figure 26 – Transfert non acquitté: diagramme d'états de l'élément demandeur.....	277
Figure 27 – Transfert non acquitté: diagramme d'états de l'élément accepteur	277
Figure 28 – Transfert acquitté: diagramme d'états de l'élément demandeur	279
Figure 29 – Transfert acquitté: diagramme d'états de l'élément accepteur	280
Figure 30 – Diagramme d'états du mécanisme de numérotation.....	282
Figure 31 – Diagramme d'états du mécanisme de nouvelles tentatives	283
Figure 32 – Diagramme d'états du mécanisme d'anticipation	286
Figure 33 – Diagrammes d'états du mécanisme de segmentation	288
Figure 34 – Diagramme d'états du mécanisme de réassemblage	291
Figure 35 – Interaction du diagramme d'états dans un transfert de données non associé	293
Figure 36 – Transfert non acquitté: diagramme d'états de l'élément demandeur	294
Figure 37 – Transfert non acquitté: diagramme d'états de l'élément accepteur	294
Figure 38 – Transfert acquitté: diagramme d'états de l'élément demandeur	296
Figure 39 – Transfert acquitté: diagramme d'états de l'élément accepteur	297
 Tableau 1 – Exemple de codage de type SEQUENCE	 173
Tableau 2 – Exemple de codage de type SEQUENCE OF	174
Tableau 3 – Exemple de codage de type CHOICE	174
Tableau 4 – Exemple de codage d'un identifiant d'objet.....	175

Tableau 5 – Exemple de codage d'une PDU	176
Tableau 6 – Types de PDU MPS.....	180
Tableau 7 – Champs d'une CompactValuePDU.....	180
Tableau 8 – Champ d'une VariableDescriptionPDU.....	187
Tableau 9 – Champ d'une AccessDescriptionPDU	189
Tableau 10 – Champ d'une TypeDescriptionPDU	190
Tableau 11 – Champ d'une ListDescriptionPDU	190
Tableau 12 – Codage des différents types de PDU MCS.....	192
Tableau 13 – Codage de la partie variable de la PDU	193
Tableau 14 – Structure de la demande d'établissement d'association	193
Tableau 15 – Structure d'une réponse d'établissement associé	197
Tableau 16 – Structure d'une demande de rupture d'association	200
Tableau 17 – Structure d'une réponse d'interruption d'association.....	200
Tableau 18 – Structure d'une demande de révocation d'association.....	201
Tableau 19 – Structure d'une demande de transfert associé	201
Tableau 20 – Structure d'un acquittement de transfert associé	202
Tableau 21 – Structure d'une demande de transfert non associé	203
Tableau 22 – Structure d'un acquittement de transfert non associé.....	204
Tableau 23 – Définitions des classes d'objets.....	206
Tableau 24 – Définition des Services de Sub MMS.....	207
Tableau 25 – Structure de la liste antiduplication.....	284
Tableau 26 – Structure de la liste de réassemblage.....	290
Tableau 27 – Valeurs du paramètre PV_RW	298
Tableau 28 – Valeurs du paramètre PV_IND.....	299
Tableau 29 – Valeurs du paramètre PV_LIS	299
Tableau 30 – Contraintes sur le paramètre PV_LIS.....	299
Tableau 31 – Valeur du paramètre PV_AT	300
Tableau 32 – Valeur du paramètre PV_RE.....	300
Tableau 33 – Valeurs du paramètre PV_UT	300
Tableau 34 – Contraintes sur le paramètre PV_RE	300
Tableau 35 – Valeurs du paramètre PH_R_A.....	300
Tableau 36 – Valeurs du paramètre PH_R_S.....	301
Tableau 37 – Valeurs du paramètre PH_R_P.....	301
Tableau 38 – Valeurs du paramètre PH_P_A	301
Tableau 39 – Valeurs du paramètre PH_P_S.....	302
Tableau 40 – Valeurs du paramètre PH_P_P.....	302
Tableau 41 – Valeurs du paramètre PH_COH.....	302
Tableau 42 – Valeurs du paramètre PH_FIA	302
Tableau 43 – Valeurs du paramètre PH_SPF.....	303
Tableau 44 – Valeurs du paramètre PH_SPM.....	303
Tableau 45 – Valeurs du paramètre PH_ACC	303
Tableau 46 – Valeurs du paramètre PH_RES.....	304
Tableau 47 – Valeurs du paramètre PH_AK.....	304

Tableau 48 – Valeurs du paramètre PH_RA.....	304
Tableau 49 – Valeurs du paramètre PH_SR.....	304
Tableau 50 – Valeurs du paramètre PH_CF	304
Tableau 51 – Contraintes sur le paramètre PH_RA.....	305
Tableau 52 – Contraintes sur le paramètre PH_SR.....	305
Tableau 53 – Valeurs du paramètre PT_OCT.....	305
Tableau 54 – Valeurs du paramètre PT_BIN	305
Tableau 55 – Valeurs du paramètre PT_VIS	306
Tableau 56 – Valeurs du paramètre PT_BOO	306
Tableau 57 – Valeurs du paramètre PT_BCD.....	306
Tableau 58 – Valeurs du paramètre PT_BTM.....	306
Tableau 59 – Valeurs du paramètre PT_INT	307
Tableau 60 – Valeurs du paramètre PT_UNO.....	307
Tableau 61 – Valeurs du paramètre PT_FPT	307
Tableau 62 – Valeurs du paramètre PT_GTM	307
Tableau 63 – Valeurs du paramètre PT_TAB	308
Tableau 64 – Valeurs du paramètre PT_STR.....	308
Tableau 65 – Contraintes sur le paramètre PT_TAB	308
Tableau 66 – Contraintes sur le paramètre PT_STR.....	308
Tableau 67 – Classes de conformité pour la gestion d'environnement.....	312
Tableau 68 – Classes de conformité pour la gestion de VMD.....	313
Tableau 69 – Classes de conformité pour la gestion de domain	314
Tableau 70 – Classes de conformité pour la Gestion de PI	315
Tableau 71 – Classes de conformité pour la gestion de variable/de variable-list	316
Tableau 72 – Classes de conformité pour la gestion d'évent.....	317

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-7: Spécifications de protocole de la couche d'application – Éléments de Type 7

AVANT-PROPOS

- 1) La Commission Électrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. À cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité National intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
- 2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
- 3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
- 4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
- 5) La CEI n'a prévu aucune procédure de marquage valant indication d'approbation et n'engage pas sa responsabilité pour les équipements déclarés conformes à une de ses Publications.
- 6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
- 7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
- 8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
- 9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de la CEI peuvent faire l'objet de droits de propriété intellectuelle ou de droits analogues. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de propriété et de ne pas avoir signalé leur existence.

NOTE L'utilisation de certains des types de protocoles est restreinte par les détenteurs des droits de propriété intellectuelle correspondants. Quoi qu'il en soit, l'engagement pris par les détenteurs, quant à une diffusion limitée desdits droits de propriété intellectuelle, permet d'utiliser un type particulier de protocole de Couche Liaison de données avec des protocoles de Couche Physique et de Couche Application dans les combinaisons de types explicitement spécifiées dans la série CEI 61784. L'utilisation des divers types de protocoles dans d'autres combinaisons peut nécessiter l'autorisation de leurs détenteurs de droits de propriété intellectuelle respectifs.

La Norme internationale CEI 61158-6-7 a été établie par le sous-comité 65C: Réseaux de communication industriels, du comité d'études 65 de la CEI: Mesure, commande et automation dans les processus industriels.

La première édition de cette norme et ses parties d'accompagnement de la série CEI 61158-6 annulent et remplacent la CEI 61158-6:2003. L'édition de cette partie constitue une révision rédactionnelle.

Cette édition de la CEI 61158-6 inclut les modifications majeures suivantes par rapport à l'édition précédente:

- a) suppression de l'ancien bus de terrain de Type 6 du fait d'un manque de pertinence pour le marché;
- b) addition de nouveaux types de bus de terrain;
- c) répartition de la partie 6 de la troisième édition en plusieurs parties numérotées -6-2, -6-3, ...

La présente version bilingue (2013-09) correspond à la version anglaise monolingue publiée en 2007-12.

Le texte anglais de cette norme est issu des documents 65C/476/FDIS et 65C/487/RVD.

Le rapport de vote 65C/487/RVD donne toute information sur le vote ayant abouti à l'approbation de cette norme.

La version française de cette norme n'a pas été soumise au vote.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de maintenance indiquée sur le site web de la CEI sous <http://webstore.iec.ch> dans les données relatives à la publication recherchée. À cette date, la publication sera:

- reconduite;
- supprimée;
- remplacée par une édition révisée, ou
- amendée.

NOTE La révision de la présente norme sera synchronisée avec les autres parties de la série CEI 61158.

La liste de toutes les parties de la série CEI 61158, publiée sous le titre général *Industrial communication networks – Fieldbus specifications* (*Réseaux de communication industriels – Spécifications de bus de terrain*), est disponible sur le site web de la CEI.

INTRODUCTION

La présente partie de la CEI 61158 s'inscrit dans une série créée pour faciliter l'interconnexion des composants de systèmes d'automatisation. Elle est liée à d'autres normes dans l'ensemble tel que défini par le modèle de référence des bus de terrain «à trois couches» décrit dans la CEI/TR 61158-1.

Le protocole d'application assure le service d'application en s'appuyant sur les services offerts par la couche de liaison de données ou une autre couche immédiatement inférieure. La présente norme a pour principal objet de fournir un ensemble de règles de communication exprimées sous la forme de procédures que doivent réaliser des entités d'application homologues (AE) au moment de la communication. Ces règles de communication visent à fournir une base saine pour le développement, dans divers buts:

- en tant que guide pour les développeurs et les concepteurs;
- dans une optique d'utilisation lors de l'essai et de l'achat de matériel;
- dans le cadre d'un accord pour l'admission de systèmes dans l'environnement de systèmes ouverts;
- en tant que précision apportée à la compréhension des communications en temps critique dans le modèle OSI.

Cette norme traite, en particulier, de la communication et de l'interfonctionnement des capteurs, effecteurs et autres appareils d'automatisation. L'utilisation conjointe de la présente norme avec d'autres normes entrant dans les modèles de référence OSI ou de bus de terrain permet à des systèmes qui ne pourraient pas, sans cela, de fonctionner ensemble dans toute combinaison.

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-7: Spécifications de protocole de la couche d'application – Éléments de Type 7

1 Domaine d'application

1.1 Généralités

La couche d'application de bus de terrain (FAL, «Fieldbus Application Layer») fournit aux programmes utilisateur un moyen d'accès à l'environnement de communication des bus de terrain. À cet égard, la FAL peut être considérée comme une «fenêtre entre les programmes d'application correspondants.»

La présente norme fournit des éléments communs pour les communications en temps critique ou non entre des programmes d'application dans un environnement et avec un matériel d'automatisation propres aux bus de terrain de Type 7. Le terme "en temps critique" est utilisé pour représenter la présence d'une fenêtre temporelle, dans le cadre de laquelle une ou plusieurs actions spécifiées doivent être réalisées avec un certain niveau de certitude défini. La non-réalisation des actions spécifiées dans le cadre de la fenêtre temporelle peut mener à l'échec des applications sollicitant les actions, avec un risque annexe pour les équipements, l'installation et éventuellement les personnes.

La présente norme définit les interactions entre les applications à distance et le comportement visible par un observateur externe assurés par la couche d'application de bus de terrain de Type 7 en termes de

- a) syntaxe abstraite définissant les unités de données de protocole de la couche d'Application, transmises entre les entités d'application en communication;
- b) syntaxe de transfert définissant les règles de codage qui sont appliquées aux unités de données de protocole de la couche d'application;
- c) diagramme d'états de contexte d'application définissant le comportement de service d'application observable entre les entités d'application en communication; et
- d) diagrammes d'états de relations entre applications définissant le comportement de communication visible entre les entités d'application en communication.

La présente norme vise à définir le protocole mis en place pour

- définir la représentation filaire des primitives de service définies dans la CEI 61158-5-7, et
- définir le comportement visible de l'extérieur associé à leur transfert.

La présente norme spécifie le protocole de la couche application de bus de terrain de Type 7, conformément au modèle de référence de base OSI (ISO/CEI 7498) et à la structure de la couche application OSI (ISO/CEI 9545).

1.2 Spécifications

L'objectif principal de la présente norme est de spécifier la syntaxe et le comportement du protocole de la couche d'application qui assure les services de la couche d'application définis dans la CEI 61158-5-7.

Un second objectif consiste à fournir des voies de migration issues des protocoles de communication industriels précédents. Ce dernier objectif génère la diversité des protocoles normalisés dans les parties de la série CEI 61158-6.

1.3 Conformité

La présente norme ne spécifie aucune mise en œuvre ou aucun produit individuel, de même qu'elle ne restreint nullement les mises en œuvre des entités de la couche d'application dans les systèmes d'automatisation industriels.

Il n'y a pas de conformité des équipements à la norme de définition du service de la couche d'application. En revanche, la conformité est obtenue par la mise en œuvre de cette spécification du protocole de la couche d'application.

2 Références normatives

Les documents de référence suivants sont indispensables pour l'application du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

CEI 60559, *Arithmétique binaire en virgule flottante pour systèmes à microprocesseurs*

IEC 61158-3-7, *Industrial communication networks – Fieldbus specifications – Part 3-7: Data-link layer service definition – Type 7 elements* (disponible en anglais uniquement)

IEC 61158-4-7, *Industrial communication networks – Fieldbus specifications – Part 4-7: Data-link layer protocol specification – Type 7 elements* (disponible en anglais uniquement)

IEC 61158-5-7, *Industrial communication networks – Fieldbus specifications – Part 5-7: Application layer service definition – Type 7 elements* (disponible en anglais uniquement)

ISO/CEI 7498-1, *Technologies de l'information – Interconnexion de systèmes ouverts – Modèle de référence de base — Partie 1. Le modèle de base*

ISO/CEI 8824, *Technologies de l'information – Interconnexion de systèmes ouverts – Spécification de la notation de syntaxe abstraite numéro 1 (ASN.1)*

ISO/CEI 8825, *Technologies de l'information – Règles de codage ASN.1: Spécification des Règles de Codage de Base, des Règles de Codage Canoniques et des Règles de Codage Distinctives*

ISO/CEI 9506-2, *Systèmes d'automatisation industrielle – Spécification de messagerie industrielle – Partie 2: Spécification de protocole*

ISO/CEI 9545, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Structure de la couche application*

3 Termes, définitions, symboles, abréviations et conventions

Pour les besoins du présent document, les définitions suivantes s'appliquent.

3.1 Termes et définitions issus des autres normes ISO/CEI

3.1.1 Termes et définitions issus de l'ISO/CEI 7498-1

- a) syntaxe abstraite
- b) entité d'application
- c) processus d'application
- d) unité de données de protocole d'application
- e) élément de service d'application
- f) invocation d'entité d'application
- g) invocation de processus d'application
- h) transaction d'application
- i) contexte de présentation
- j) système ouvert réel
- k) syntaxe de transfert

3.1.2 Termes et définitions issus de l'ISO/CEI 9545

- a) association d'application
- b) contexte d'application
- c) nom de contexte d'application
- d) invocation d'entité d'application
- e) type d'entité d'application
- f) invocation de processus d'application
- g) type de processus d'application
- h) élément de service d'application
- i) élément de service de contrôle d'application

3.1.3 Termes et définitions issus de l'ISO/CEI 8824

- a) identifiant d'objet
- b) type
- c) valeur
- d) type simple
- e) type structuré
- f) type de composant
- g) marqueur
- h) type boolean
- i) vrai
- j) faux
- k) type integer
- l) type bitstring
- m) type octetstring
- n) type null
- o) type sequence
- p) type sequence of
- q) type choice
- r) type tagged
- s) tout type
- t) module
- u) production

3.1.4 Termes et définitions issus de l'ISO/CEI 8825

- a) codage (d'une valeur de données)
- b) valeur de données
- c) octets d'identification (la forme du singulier est utilisée dans la présente norme)
- d) octet(s) de longueur (les formes du singulier et du pluriel sont utilisées dans la présente norme)
- e) octets de contenu

3.2 Termes et définitions issus de la CEI 61158-5-7

- a) relation entre applications
- b) trajet d'acheminement
- c) client
- d) AR dédiée
- e) AR dynamique
- f) classe d'erreur
- g) code d'erreur
- h) nom
- i) identifiant numérique
- j) homologue
- k) extrémité d'AR prédéfinie
- l) extrémité d'AR préétablie
- m) éditeur
- n) abonné

3.3 Termes et définitions supplémentaires

3.3.1

allouer

prendre une ressource dans une zone commune et affecter la ressource en question à l'usage exclusif d'une entité spécifique

3.3.2

application

fonction ou structure de données pour laquelle des données sont consommées ou produites

3.3.3

objets d'application

plusieurs classes d'objet qui gèrent et permettent l'échange de messages lors du temps d'exécution sur le réseau et au sein de l'appareil du réseau

3.3.4

attribut

description d'une caractéristique ou d'une fonction extérieurement visible d'un objet

NOTE Les attributs d'un objet contiennent les informations concernant les parties variables d'un objet. Généralement, ces attributs fournissent des informations relatives au statut ou régissent le fonctionnement d'un objet. Les attributs peuvent également altérer le comportement d'un objet. Ces attributs sont répartis en attributs de classe et d'instance.

3.3.5

comportement

indication de la manière dont un objet répond à des événements particuliers

3.3.6

appelé

utilisateur ou fournisseur de services qui reçoit une primitive d'indication ou une APDU de demande

3.3.7

appelant

utilisateur ou fournisseur de service qui déclenche une primitive de demande ou une APDU de demande

3.3.8

classe

ensemble d'objets, qui représentent tous le même type de composant de système

NOTE Une classe est une généralisation d'un objet, un modèle pour définir des variables et des méthodes. Tous les objets d'une classe ont une forme et un comportement identiques, mais leurs attributs contiennent généralement des données différentes.

3.3.9

attributs de classe

attribut partagé par tous les objets de la même classe

3.3.10

code de classe

identifiant unique affecté à chaque classe d'objets

3.3.11

service spécifique à la classe

service défini par une classe d'objets particulière afin d'exécuter une fonction requise non exécutée par un service commun

NOTE Un objet spécifique à la classe est unique à la classe d'objets qui le définit.

3.3.12

client

- a) objet qui utilise les services d'un autre objet (serveur) pour accomplir une tâche
- b) initiateur d'un message auquel un serveur réagit

3.3.13

horloge

appareil fournissant une mesure du temps écoulé depuis un temps de référence défini

NOTE Il existe deux types d'horloges définis dans la CEI 61588, les horloges limites et les horloges ordinaires.

3.3.14

objets de communication

composants qui assurent la gestion et la réalisation d'un échange lors du temps d'exécution de messages sur le réseau

EXEMPLES Objet Connection Manager (Gestionnaire de Connexion), objet Unconnected Message Manager (Gestionnaire de Message non Connecté) (UCMM) objet Message Router (Routeur de Message).

3.3.15

connexion

liaison logique entre des objets d'application qui peuvent être au sein du même appareil ou dans des appareils différents

NOTE Les connexions peuvent être point à point ou multipoint.

3.3.16

identifiant de connexion (CID)

identifiant attribué à une émission associée à une connexion particulière entre producteurs et consommateurs, fournissant un nom pour une information d'application spécifique

3.3.17

point de connexion

tampon qui est représenté comme une sous-instance d'un objet Assembly (Assemblage)

3.3.18

consommer

acte consistant à recevoir des données d'un producteur

3.3.19

consommateur

nœud ou puits qui reçoit des données d'un producteur

3.3.20

application consommatrice

application qui consomme des données

3.3.21

cyclique

répétitif d'une manière régulière

3.3.22

appareil

matériel physique connecté à la liaison

NOTE Un appareil peut contenir plus d'un nœud.

3.3.23

profil d'appareil

ensemble d'informations et de fonctionnalités dépendantes de l'appareil fournissant une cohérence entre des appareils similaires du même type

3.3.24

nœud d'extrémité

nœud producteur ou consommateur

3.3.25

point d'extrémité

une des entités en communication impliquées dans une connexion

3.3.26

erreur

discordance entre une valeur ou un état calculé(e), observé(e) ou mesuré(e) et la valeur ou l'état spécifié(e) ou théoriquement correct(e)

3.3.27

trame

synonyme déconseillé de DLPDU

3.3.28

instance

occurrence physique réelle d'un objet au sein d'une classe qui identifie l'un des plusieurs objets au sein de la même classe d'objets

EXEMPLE «California» (Californie) est une instance de l'état de la classe d'objets.

NOTE Les termes objet, instance et instance d'objet sont utilisés pour faire référence à une instance spécifique.

3.3.29

attribut d'instance

attribut qui est propre à une instance d'objet et qui n'est pas partagé par la classe d'objets

3.3.30

instancié

objet qui a été créé dans un appareil

3.3.31

interopérabilité

capacité des entités de la Couche Utilisateur à effectuer des opérations coordonnées et coopératives en utilisant les services de la FAL

3.3.32**little-endian (petit-boutiste)**

décrit un modèle d'organisation de mémoire qui stocke l'octet de poids faible à l'adresse la plus petite, ou pour le transfert, qui transfère l'octet de poids le plus faible en premier

3.3.33**information de gestion**

information accessible sur le réseau qui vient à l'appui de la gestion de l'environnement des bus de terrain

3.3.34**membre**

élément d'un attribut qui est structuré comme étant un élément d'une matrice

3.3.35**routeur de message**

objet au sein d'un nœud qui distribue les demandes de messagerie vers les objets d'application appropriés

3.3.36**connexion multipoint**

connexion d'un nœud à plusieurs autres nœuds

NOTE Les connexions multipoint permettent aux messages d'un seul producteur d'être reçus par plusieurs nœuds consommateurs.

3.3.37**réseau**

ensemble de nœuds reliés par un certain type de support de communication, notamment des répéteurs intermédiaires, des ponts, des routeurs et des passerelles de couche inférieure

3.3.38**objet**

représentation abstraite d'un composant particulier au sein d'un appareil, généralement un ensemble de données (sous la forme de variables) et de méthodes (procédures) connexes pour effectuer des opérations sur les données en question qui ont une interface et un comportement clairement définis

3.3.39**service spécifique à l'objet**

service propre à la classe d'objets qui le définit

3.3.40**initiateur**

client qui est chargé d'établir un chemin de connexion vers la cible

3.3.41**connexion point à point**

connexion qui existe entre exactement deux objets d'application

3.3.42**produire**

acte consistant à envoyer des données devant être reçues par un consommateur

3.3.43**producteur**

nœud qui est chargé d'envoyer des données

3.3.44

réception

utilisateur de services qui reçoit une primitive confirmée ou une primitive non confirmée, ou un fournisseur de services qui reçoit une APDU confirmée ou une APDU non confirmée

3.3.45

ressource

une ressource est la capacité de traitement ou d'information d'un sous-système

3.3.46

envoi

utilisateur de services qui envoie une primitive confirmée ou une primitive non confirmée, ou un fournisseur de services qui envoie une APDU confirmée ou une APDU non confirmée

3.3.47

service

opération ou fonction accomplie par un objet et/ou une classe d'objets, à la demande d'un autre objet et/ou d'une autre classe d'objets

3.4 Abréviations et symboles

ASCII	Code américain normalisé pour l'échange d'information (American Standard Code for Information Interchange)
CID	connection ID (identifiant de connexion)
DLL	data- link layer (couche de liaison de données)
PDU	protocol data unit (unité de données de protocole)
OSI	open systems interconnection (interconnexion de systèmes ouverts) (voir ISO/CEI 7498)
Rcv	receive (réception)
Rx	réception
SDU	service data unit (unité de données de service)
STD	state transition diagram (diagramme des transitions d'état), utilisé pour décrire le comportement d'un objet
TPDU	transport protocol data unit (unité de données de protocole de transport)
Tx	émission
Xmit	émission

3.5 Conventions

3.5.1 Concept général

La FAL est définie comme un ensemble d'ASE orientée objet. Chaque ASE est spécifié dans un paragraphe séparé. Chaque spécification d'ASE est composée de trois parties: ses définitions de classe, ses services et sa spécification de protocole. Les deux premières parties sont contenues dans la CEI 61158-5. La spécification de protocole pour chacun des ASE est définie dans la présente norme.

Les définitions de classe définissent les attributs des classes pris en charge par chaque ASE. Les attributs sont accessibles à partir des instances de la classe utilisant les services d'ASE Management (Gestion) spécifiés dans la CEI 61158-5. La spécification des services définit les services fournis par l'ASE.

La présente norme utilise les conventions descriptives données dans l'ISO/CEI 10731.

3.5.2 Conventions pour les définitions de classe

Les définitions du mapping aux couches de liaison de données sont décrites à l'aide de modèles. Chaque modèle est constitué d'une liste d'attributs de la classe. La forme générale du modèle est définie dans la CEI 61158-5.

3.5.3 Conventions de syntaxe abstraite

Lorsque le paramètre «optionalParametersMap» est utilisé, un nombre de bits correspondant à chaque production OPTIONAL ou DEFAULT est donné sous forme de commentaire.

3.6 Conventions utilisées dans les diagrammes d'états

3.6.1 Généralités

Ce paragraphe rappelle les principales définitions associées aux trois techniques de spécification utilisées: 'Réseaux d'Évaluation', 'Objets Abstraits', 'Conventions de Service'.

3.6.2 Réseaux d'évaluation

Pour être utile au lecteur, certains mécanismes sont expliqués par une représentation graphique comme le montre la Figure 1. Ces représentations sont basées sur le réseau d'évaluation représenté sur un graphique orienté qui possède trois types de nœuds:

- les états, représentés par un disque,
- les demandes représentées par un rectangle,
- les transitions, représentées par une ligne horizontale.

Le réseau d'évaluation est construit conformément au principe suivant:

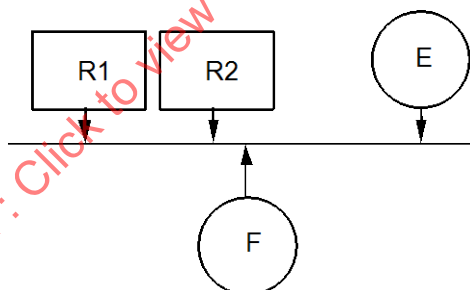


Figure 1 – Exemple de réseau d'évaluation

Lorsque le système décrit est dans l'état E, l'arrivée des demandes R1 ou R2 se traduit par une transition qui le commute dans l'état F.

Par ailleurs, le franchissement d'une transition a lieu, par convention, dans le temps zéro.

3.6.3 Actions simples

Des actions simples peuvent être associées à chaque transition, correspondant soit à l'émission des demandes ou à l'exécution d'actions locales.

Elles sont matériellement représentées par un triangle, étiqueté avec le nom de la demande ou de la description de l'action exécutée.

3.6.4 Conditions

Le franchissement d'une transition peut concerner un état. Dans ce cas, le graphe orienté associé à la représentation montre un état associé à l'arc qui précède la transition.

Un corollaire de cette représentation permet de définir des choix pour le franchissement d'une transition à partir du même état initial.

De la même manière que pour une transition simple, l'émission d'une action peut être associée au franchissement d'une transition conditionnelle.

4 Syntaxe abstraite de type de données

4.1 Spécification relative à la syntaxe abstraite de données

La présente norme utilise les termes suivants définis dans la syntaxe abstraite numéro 1 (ASN1). Les autres définitions ou les définitions ajoutées sont les suivantes:

4.1.1 Boolean

Une valeur de type Boolean est codée sur un octet dont tous les bits sont nuls pour FALSE; toute autre combinaison signifie TRUE.

Exemple de notation du type:

Response ::= BOOLEAN

Exemple de notation et de codage des valeurs:

Response value ::= TRUE

la valeur est codée, par exemple, FFh ou 6Dh.

4.1.2 Integer

La valeur de type INTEGER est codée comme une séquence de M octets décrivant un nombre binaire:

- soit en tant que complément à 2 si les valeurs peuvent être positives et négatives: le bit de poids fort et le bit de poids faible sont respectivement le bit 8 du premier octet et le bit 1 du dernier octet. Le bit 8 du premier octet représente le signe de l'entier. Si la taille représente le nombre de bits de l'entier (c'est-à-dire $\text{taille} = M \times 8$), la valeur de l'entier se situe entre $-2^{**}(\text{taille}-1)$ et $+2^{**}(\text{taille}-1)-1$;

Exemple de notation du type:

Signed8 ::= INTEGER (-128..127)

Exemple de notation et de codage des valeurs:

integer Signed8 ::= -128

L'entier est codé: FFh.

- ou en tant que valeur absolue si les valeurs sont toutes positives ou négatives: le bit de poids fort et le bit de poids faible sont respectivement le bit 8 du premier octet et le bit 1 du dernier octet. Si la taille du nombre de bits de l'entier (c'est-à-dire, $\text{taille} = M \times 8$), la valeur de l'entier se situe entre $-2^{**}(\text{taille})$ et $+2^{**}(\text{taille})-1$.

Exemple de notation du type:

Unsigned8 ::= INTEGER (0..255)

Exemple de notation et de codage des valeurs:

integer Unsigned8 ::= 255

L'entier est codé: FFh.

4.1.3 Bit string

La valeur de type BIT STRING est codée sur une séquence de M octets; le premier bit de la chaîne binaire est aligné avec le bit 8 de l'octet de poids faible de la séquence d'octets et son numéro est zéro. Le nombre de bits inutilisé doit être compris entre les bits 1 et 7 de l'octet de poids fort de la séquence d'octets. La vraie valeur d'un bit de la chaîne binaire prend la valeur binaire 1; les bits inutilisés peuvent prendre la valeur 0 ou 1.

Exemple de notation du type:

Bitstring ::= BIT STRING SIZE(32)

Exemple de notation et de codage des valeurs:

binaryvalue Bitstring ::= '5F291CD0'H

la valeur binaire est codée: 5Fh 29h 1Ch D0h

4.1.4 Octet string

La valeur de type OCTET STRING est codée sur une séquence de M octets; le premier octet de type OCTET STRING est aligné avec l'octet de poids faible de la séquence d'octets.

Exemple de notation du type:

OctetString ::= OCTET STRING SIZE(4)

Exemple de notation et de codage des valeurs:

octetvalue OctetString ::= '5F291CD0'H

la valeur octetvalue est codée: 5Fh 29h 1Ch D0h.

4.1.5 Sequence (sequence ou sequence of)

Le codage de type SEQUENCE ou de type SEQUENCE OF est la juxtaposition du codage de l'élément qui le constitue.

Exemple de notation du type:

INFO1 ::= SEQUENCE {name String, ok BOOLEAN}

String ::= OCTET STRING SIZE(5) –chaîne de 5 octets

Exemple de notation et de codage des valeurs:

value INFO1 ::= {name "SMITH", ok TRUE}

la valeur est codée comme indiqué dans le Tableau 1.

Tableau 1 – Exemple de codage de type SEQUENCE

0006h	53h	4Dh	49h	54h	48h	FFh
Longueur de SEQUENCE	"S"	"M"	"I"	"T"	"H"	TRUE

La ligne du haut représente le codage propre et celle du bas décrit ce qui est codé:

Exemple du type:

INFO2 ::= SEQUENCE OF date

date ::= OCTET STRING SIZE (8) --YYYYMMDD

Exemple de notation et de codage des valeurs:

value INFO2::= { { date "19571111"}, {date "19590717"}}

la valeur est codée comme indiqué dans le Tableau 2.

Tableau 2 – Exemple de codage de type SEQUENCE OF

00h	10h				
nombre	d'octets				
31h	39h	35h	37h	31h	31h
"1"	"9"	"5"	"7"	"1"	"1"
31h	31h	31h	39h	35h	39h
"1"	"1"	"1"	"9"	"5"	"9"
30h	37h	31h	37h		
"0"	"7"	"1"	"7"		

Dans chaque paire de lignes, la ligne du haut représente le codage propre et celle du bas décrit ce qui est codé. Chaque tableau est lu de gauche à droite.

4.1.6 Choice

Le codage de type CHOICE est le codage de la possibilité sélectionnée dans ce CHOICE (CHOIX).

Exemple de notation du type:

INFO::= CHOICE {name [0] String, age [1] Unsigned8}

String::= OCTET STRING SIZE(5) –chaîne de 5 octets

Unsigned8::= INTEGER (0..127) – entier non signé de 8 bits

Exemple de notation et de codage des valeurs:

value INFO::= {name "SMITH"}

la valeur est codée comme indiqué dans le Tableau 3.

Tableau 3 – Exemple de codage de type CHOICE

80h	53h	4Dh	49h	54h	48h
identification	"S"	"M"	"I"	"T"	"H"

La ligne du haut représente le codage propre et celle du bas décrit ce qui est codé.

4.1.7 Null

Aucun codage du contenu n'est associé à l'élément null.

Le codage d'identification est possible si NULL est une possibilité d'un choix (voir 10.1.2.5.2).

Exemple de notation du type:

```

Type-Room ::= SEQUENCE {
    number Unsigned8,
    person CHOICE {
        name [0] String,
        default [1] NULL --espace non utilisé
    }
}

```

Unsigned8 ::= INTEGER (0..127)

String ::= OCTET STRING SIZE(12)

Exemple de notation et de codage des valeurs:

value-Room Type-Room ::= {number 30h, person { default NULL}}

La valeur Room est codée 0002h 30h 81h

Le codage d'une longueur est obligatoire si NULL est facultatif (dans ce cas, la longueur est nécessairement égale à zéro).

4.1.8 Identifiant d'objet

Le codage est constitué d'une liste ordonnée de codage de sous-identifiants concaténés.

Chaque sous-identifiant est représenté par une séquence d'un ou de plusieurs octets. Chaque octet est codé comme suit.

- Le bit 8 de chaque octet indique s'il est le dernier de la séquence.
- Le bit 8 du dernier octet est mis à un.
- Le bit 8 de chaque octet précédent est mis à zéro.
- Ensemble, les bits 7 à 1 des octets de la séquence codent le sous-identifiant, en sachant que:

Les bits 5, 6 et 7 valent 0.

- Les bits 1 à 4 sont utilisés pour coder chaque chiffre de chaque sous-identifiant. Le codage se produit sous la forme d'un entier binaire non signé.

Exemple: {iso standard 9506 part(2) mms-general-module-version1(2)} est codé comme présenté dans le Tableau 4.

Tableau 4 – Exemple de codage d'un identifiant d'objet

81h	80h	09h	05h	00h
"iso"	"standard"	9	5	0
86h	82h	82h		
6	part(2)	mms...(2)		

Dans chaque tableau, la ligne du haut représente le codage propre et celle du bas décrit ce qui est codé. Chaque tableau est lu de gauche à droite.

4.1.9 Default

Le mot-clé DEFAULT est ignoré.

4.2 Syntaxe abstraite de PDU FAL

Ceparagraphe illustre les règles de codage FER spécifiées dans la présente norme, en montrant la représentation en octets d'un élément défini dans l'ASN.1 DES BUS DE TERRAIN.

```

PDU ::= CHOICE {
    req [0] IMPLICIT PDUreq,
    rep [1] IMPLICIT PDUrep, ....
}

PDUrep ::= SEQUENCE {
    invokeID Unsigned32,
    Response }

Response ::= CHOICE {
    status                                [0] IMPLICIT StatusResponse, ...
    getprog                              [45] IMPLICIT GetProgramInvocationAttributesResponse, ...}

GetProgramInvocationAttributesResponse ::= SEQUENCE {
    pi_state                             [0] IMPLICIT PiState,
    listofDomainId                       [1] IMPLICIT ListOfObjectID,
    mmsdeletable                         [2] IMPLICIT BOOLEAN,
    reusable                             [3] IMPLICIT BOOLEAN,
    monitor                              [4] IMPLICIT BOOLEAN,
    executionargument                    [5] IMPLICIT OCTET STRING OPTIONAL
}

PiState ::= Unsigned8

Unsigned8 ::= INTEGER (0..127) –entier non signé de 8 bits

ListOfObjectID ::= SEQUENCE OF Identifier

Identifier ::= Unsigned16

Unsigned16 ::= INTEGER (0..32767) –entier non signé de 16 bits

Unsigned32 ::= INTEGER(0..2147483647) –entier non signé de 32 bits

La valeur enregistrée est la suivante:
{ rep {invokeID 1,
    getprog { pi-state 8,
    listofDomainId { {1024}, {1025}, {1026}, {1027} },
    mmsdeletable TRUE,
    reusable FALSE,
    monitor TRUE,
    executionargument "ARGUMENT"}
}

```

Sa représentation est comme suit: le codage des octets est décrit avec les tableaux à deux lignes présentés ci-dessous dans lesquels la ligne du haut représente le codage propre et celle du bas décrit ce qui est codé. Les tableaux sont lus de gauche à droite. Voir Tableau 5 pour plus de détails.

Tableau 5 – Exemple de codage d'une PDU

81h	00h	00h	00h	01h	ADh
REP	invokeID	(32 b ...		...)	getprog

08h	00h	08h	04h	00h	04h	01h
pi-state	number	of octets	1024	(16 b)	1025	(16b)

04h	02h	04h	03h	FFh	00h	FFh	00h
1026	(16b)	1027	(16b)	TRUE	FALSE	TRUE	number

08h	41h	52h	47h	55h	4Dh	45h	4Eh
of octets	"A"	"R"	"G"	"U"	"M"	"E"	"N"

54h
"T"

5 Syntaxe de transfert

5.1 Codage compact

Les règles de codage appliquées aux valeurs des types de PDU représentées en utilisant la notation de syntaxe abstraite ASN1 (ISO 8824) donnent la syntaxe de transfert.

Dans l'environnement de MPS, les valeurs associées aux types de PDU, sont globalement codées conformément aux règles de codage de l'ASN1 (ISO/CEI 8825), avec quelques restrictions.

Les règles de codage qui donnent la syntaxe de transfert de MPS associent un ensemble d'octets à chacun des différents types de PDU.

Cette séquence d'octets comprend trois composantes:

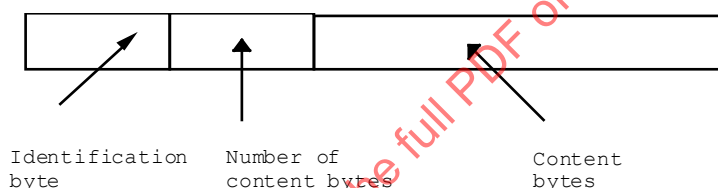
- identification
- taille
- contenu.

Dans le cas général, les règles de codage autorisent un contenu qui se structure de manière récursive en termes d'Identification, de Taille et de Contenu.

NOTE Dans le cas d'une PDU de type CompactValue, le codage a un seul niveau.

L'organisation d'un niveau de codage correspond à celui de l'ISO/CEI 8825.

La structure du codage est indiquée à la Figure 2.



Légende

Anglais	Français
Identification byte	Octet d'identification
Number of content bytes	Nombre d'octets de contenu
Content bytes	Octets de contenu

Figure 2 – Codage d'une CompactValue

Les restrictions suivantes relatives au codage par rapport à la norme de MPS s'appliquent:

La composante «identification» est codée sur 1 octet.

La composante «contenu» est codé sur un maximum de 126 octets.

5.2 Codage de type de données

5.2.1 Types de données de la PDU FAL ASE MPS

5.2.1.1 Résumé

Les spécifications du protocole fournissent un ensemble de règles de communication exprimées en termes d'échanges de données et de procédures qui doivent être pris en charge par des entités d'application.

La conformité AL TYPE 7 est basée sur le respect de ces règles de communication qui sont des lignes directrices pour les essais de conformité afin de garantir l'interopérabilité.

Cette spécification de protocole est constituée de:

- la description des différentes unités de données de protocole (PDU) échangées entre les entités en communication. Cette description est affectée par l'utilisation d'une syntaxe abstraite ;
- les règles de codage qui transforment la syntaxe abstraite en syntaxe de transfert ;
- la description des procédures de protocole en termes d'interactions entre les primitives de chacun des services d'application et les primitives de service de liaison de données associées.

NOTE Dans l'environnement de MPS, il n'y a pas d'interaction directe entre une entité d'application producteur et une entité consommateur au cours des échanges liés à une primitive de service.

La spécification des A_PDU dans l'environnement de MPS est réalisée avec une notation définie indépendamment du codage. Cette notation constitue la syntaxe abstraite.

La syntaxe abstraite est ensuite transformée par un ensemble de règles de codage qui sont appliquées aux différents types d'A_PDU. Ces règles de codage donnent la syntaxe de transfert.

La conjonction de la syntaxe abstraite et des règles de codage spécifie le contenu des octets échangés entre les entités d'application en utilisant les services de liaison de données.

NOTE La représentation réelle au niveau de l'interface entre l'utilisateur et une entité d'application ne relève pas du domaine d'application de la présente norme et dépend de la mise en œuvre.

5.2.1.2 Notation de la syntaxe

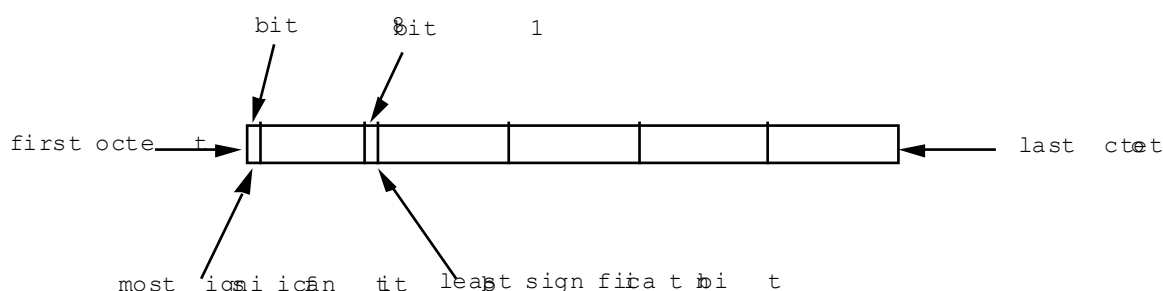
La spécification des différentes A_PDU associées à des services de MPS est effectuée à l'aide de la notation de syntaxe abstraite ASN.1 définie dans l'ISO/CEI 8824.

Toutes les définitions ASN1 de la présente norme font partie du module: « AL TYPE 7 - -MPS-1 ». Tous les éléments de grammaire qui constituent le module Part5-3-MPS-1 correspondent à la première version de MPS.

Déclaration du module AL TYPE 7 - -MPS-1:
AL TYPE 7 - -MPS-1 DEFINITIONS:= BEGIN

NOTE Afin de répondre aux exigences de la notation ASN1, tous les mots-clés du langage sont écrits en majuscules et les noms des symboles finaux commencent par une majuscule. En outre, tous les éléments de grammaire existants dans la présente norme qui participent à la définition du module type Al 7 - MPP-1, sont indiqués avec une double ligne avant et après.

Citation générale: La présente norme spécifie les bits de chaque octet d'un codage en utilisant les expressions 'bit de poids fort' et 'bit de poids faible'. Les bits d'un octet sont numérotés de 8 à 1: le bit 8 étant 'le bit de poids fort' et le bit 1 'le bit de poids faible'. Ce codage est résumé à la Figure 3.



Légende

Anglais	Français
First octet	Premier octet

Anglais	Français
Most significant bit	Bit de poids fort
Least significant bit	Bit de poids faible
Last octet	Dernier octet

Figure 3 – Organisation des bits et des octets au sein d'une PDU

5.2.1.3 Types prédéfinis simples

La spécification de la PDU utilise des types prédéfinis simples.

Certains types simples sont prédéfinis par le biais d'une déclaration qui définit des limites pour les valeurs de ces types.

```
Unsigned4      ::= INTEGER          -- valeurs limitées entre 0 et 15
Unsigned8      ::= INTEGER          -- valeurs limitées entre 0 et 255
Unsigned16 ::= INTEGER              -- valeurs limitées entre 0 et 65535
Symbol ::= Visible String           -- nombre de caractères limité entre 0 et 16
```

Dans l'environnement de MPS, les noms de variables, les noms d'accès, les noms de types et les noms de listes sont décrits comme une combinaison de longueur maximale de 16 caractères.

Les caractères appartiennent à l'ensemble défini pour le type 'visible string' de la notation de syntaxe ASN1 (ISO 8824).

Dans cet ensemble, les caractères suivants sont autorisés:

```
Lettres majuscules      (A..Z)
Lettres                  (a..z)
Chiffres                  (0..9)
Trait bas                (_)
Symbole monétaire        ($)
```

Le caractère 'espace' n'est pas autorisé.

Les chaînes peuvent ne pas commencer par un nombre.

Les types 'Unsigned16', 'Unsigned8' et 'Unsigned4' sont utilisés dans la présente norme pour décrire les limites des valeurs maximales des types de MPS.

5.2.1.4 Types de PDU

5.2.1.4.1 Définition de premier niveau

Les PDU spécifiées dans l'environnement de MPS acheminent les informations associées aux objets *Variable*. Le type de PDU utilisé dépend directement de la valeur des attributs *Construction* et *A_Name* de l'instance du *Type constructor* associé à la variable et est résumé dans le Tableau 6.

Tableau 6 – Types de PDU MPS

Construction		Type de PDU
SIMPLE		CompactValuePDU
ARRAY		CompactValuePDU
STRUCTURE		CompactValuePDU
PRÉDÉFINIE	A_Name = K_DVAR	VariableDescriptionPDU
	A_Name = K_DACCESS	AccessDescriptionPDU
	A_Name = K_DTYPE	TypeDescriptionPDU
	A_Name = K_DLIST	ListDescriptionPDU
EXPLICITE		ExplicitPDU

Le codage des valeurs de ces différents types de PDU utilise la syntaxe de transfert associée à ASN1 partie 2 (ISO 8825). Néanmoins, les règles de compactage et de compression de tableau peuvent être mises en œuvre pour optimiser le débit sur le bus.

NOTE Ces règles de compactage permettent la représentation des variables de type structuré en utilisant une chaîne d'octets.

Spécification du type de PDU utilisé dans MPS:

MPS-pdu:: = CHOICE

```
{
  CompactedVariableValue      [APPLICATION 0]  IMPLICIT CompactValuePDU,
  ExplicitVariableValue       [APPLICATION 1]  IMPLICIT ExplicitPDU,
  VariableDescription         [APPLICATION 2]  IMPLICIT VariableDescriptionPDU,
  AccessDescription           [APPLICATION 3]  IMPLICIT AccessDescriptionPDU,
  TypeDescription             [APPLICATION 4]  IMPLICIT TypeDescriptionPDU,
  ListDescription             [APPLICATION 5]  IMPLICIT ListDescriptionPDU,
  extensions                   [APPLICATION 6]  ANY
}
```

NOTE Les PDU de type [APPLICATION 7] à [APPLICATION 15] sont réservées par la présente norme à une future définition d'autres types de PDU. Les PDU de type [APPLICATION 16] à [APPLICATION 30] sont réservées à l'utilisation de la norme de Gestion de Réseau. Par ailleurs, la PDU de type ANY est utilisée pour permettre la définition future de types de PDU supplémentaires dans les normes d'accompagnement.

5.2.1.4.2 PDU VariableCompactValue

5.2.1.4.2.1 Généralités

Cette PDU représente la valeur des variables de type de construction SIMPLE, STRUCTURE ou ARRAY, et éventuellement le statut de production associé. Ces types sont utilisés pour les variables de la classe NORMAL ou SYNCHRONIZATION.

La PDU de type CompactValuePDU est représentée à l'aide du type OCTET STRING:

Compact-value-pdu:: = OCTET STRING

-- La longueur totale de la PDU ne doit pas dépasser 128 octets.

NOTE Le champ de cette PDU permet le codage des attributs suivants dans les objets de la couche d'application de MPS. Voir Tableau 7.

Tableau 7 – Champs d'une CompactValuePDU

Champ	Classe	Attribut
Compact_value_PDU	Variable	Public value Transmitted status value

Le type OCTET STRING d'ASN1 sera ensuite codé conformément aux règles de codage ASN1 (ISO 8825) comme spécifié dans le chapitre relatif à la syntaxe de transfert.

NOTE La représentation d'une variable de MPS de type OCTET STRING d'ASN1 n'est possible qu'en tenant compte des restrictions suivantes.

- Les producteurs et les consommateurs de la variable ont une connaissance préalable du type de MPS de la variable, et n'ont pas besoin de transmettre la description du type avec la valeur de la variable. La connaissance est disponible par l'accès à la variable de description et aux variables des types de description.
- Le type de la variable a une longueur fixe et ne comporte aucun champ conditionnel ou facultatif.

La représentation de la valeur publique associée à une variable par l'utilisation des règles de compactage est effectuée à partir du premier octet de contenu de CompactValuePDU.

Les statuts de production sont représentés sur le dernier octet lorsque cette option est sélectionnée.

La valeur de l'octet de longueur de la PDU représente le nombre d'octets utilisé pour coder la valeur de la variable et l'octet du statut de production quand il existe.

La représentation du statut de production sur cet octet est effectuée comme suit.

- Le statut de rafraîchissement est codé sur le bit 1. Ce bit est mis à '1' lorsque le statut de rafraîchissement est 'TRUE', et à '0' lorsqu'il est 'FALSE'.
- Le rafraîchissement ponctuel est codé sur le bit 2. Ce bit est mis à '1' lorsque le rafraîchissement ponctuel est 'TRUE', et à '0' lorsqu'il est 'FALSE'.
- Les bits 3 à 8 sont réservés.

Le codage d'une variable donnée est représenté avec un nombre d'octets constant issu de la spécification du type de cette variable.

5.2.1.4.2.2 Règle de codage des valeurs de type BOOLEAN

Le codage du type boolean est effectué sur un seul octet.

Si la valeur est

FALSE: l'octet est 00_{16}

TRUE: l'octet peut prendre n'importe quelle valeur différente de zéro, au choix de l'entité producteur.

5.2.1.4.2.3 Règle de codage des valeurs de type INTEGER

Le codage d'une valeur de type integer est réalisé sur un ou plusieurs octets. Le nombre maximal d'octets est fixé par la taille maximale autorisée pour le type INTEGER, soit 32 octets.

Le nombre d'octets utilisé pour coder un type INTEGER correspond à $E[(e-1)/8] + 1$ où E est la partie entière et e la valeur de l'attribut size associée à ce type.

Les octets de contenu doivent être un nombre binaire en complément à deux égal à la valeur entière, et être composés des bits 8 à 1 du premier octet, suivis des bits 8 à 1 du deuxième octet, et ainsi de suite jusques et y compris le dernier des octets de contenu.

NOTE La valeur d'un nombre binaire en complément à deux est obtenue en numérotant les bits des octets de contenu à partir du bit 1 du dernier octet jusqu'au bit 8 du premier octet. À chaque bit est affectée une valeur numérique de 2^{n-1} où n est la position du bit dans la séquence de numérotation. La valeur d'un nombre binaire en

complément à deux est obtenue en faisant la somme des valeurs numériques affectées à chacun des bits égaux à '1' (sauf le bit 8 du premier octet), de laquelle est issue la valeur signée du bit 8 du premier octet. Le signe est négatif lorsque le bit 8 du premier octet est mis à '1'.

5.2.1.4.2.4 Règle de codage des valeurs de type BITSTRING

Le codage d'une valeur de type bitstring est réalisé sur un ou plusieurs octets comme illustré à la Figure 4.

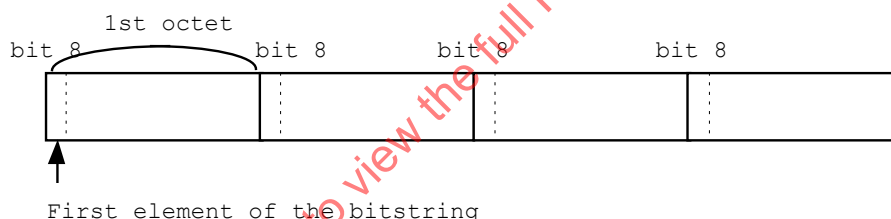
Le nombre maximal d'octets est fixé par la taille maximale autorisée pour un type BITSTRING, soit 32 octets.

Le nombre d'octets utilisé pour coder un type BITSTRING correspond à $E[(e-1)/8] + 1$ où E est la partie entière et e la taille de ce type.

Les bits de la chaîne binaire, qui commence à partir du premier bit et se termine par le dernier bit, doivent être placés dans la séquence et dans cet ordre:

- dans les bits 8 à 1 du premier octet pour les 8 premiers bits de la chaîne,
- dans les bits 8 à 1 du deuxième octet pour les 8 bits suivants,
- puis dans les bits 8 à 1 de chacun des octets suivants,
- suivis d'autant de bits que nécessaire dans le dernier octet, en commençant toujours par le bit 1.

Les bits inutilisés du dernier octet ont une valeur choisie par l'entité producteur.



Légende

Anglais	Français
1 st octet	1er octet
First element of the bitstring	Premier élément de la chaîne binaire

Figure 4 – Codage de type bitstring

5.2.1.4.2.5 Règle de codage des valeurs de type UNSIGNED

Le codage d'une valeur de type unsigned integer est réalisé sur un ou plusieurs octets. Le nombre maximal d'octets est fixé par la taille maximale autorisée pour le type UNSIGNED, soit 32 octets. Le nombre d'octets utilisé pour coder un type UNSIGNED correspond à $E[(e-1)/8] + 1$ où E est la partie entière et e la valeur de l'attribut size associée à ce type.

Les octets de contenu doivent être un nombre binaire équivalent à la valeur de type integer, et être composés des bits 8 à 1 du premier octet, suivis des bits 8 à 1 du deuxième octet, et ainsi de suite jusques et y compris le dernier des octets de contenu.

NOTE La valeur d'un nombre binaire est obtenue en numérotant les bits des octets de contenu à partir du bit 1 du dernier octet jusqu'au bit 8 du premier octet. À chaque bit est affectée une valeur numérique de 2^{n-1} où n est la position du bit dans la séquence de numérotation.

5.2.1.4.2.6 Règle de codage des valeurs de type OCTETSTRING

Le codage d'une valeur de type OCTETSTRING contient zéro, un ou plusieurs octets de contenu dont la valeur est égale à celle des octets de la valeur des données, et sont placés dans le même ordre que ceux-ci, le bit de poids fort d'une valeur de données étant aligné avec le bit de poids fort d'un octet de contenu.

Le nombre d'octets utilisé pour coder un type OCTETSTRING correspond directement au nombre de termes de cette chaîne dans la limite de 126 octets.

5.2.1.4.2.7 Règle de codage des valeurs de type VISIBLE STRING

La valeur d'une chaîne visible est composée d'une chaîne de caractères de l'ensemble spécifié dans l'ISO 8824. Chacun des caractères de cette chaîne est codé en un octet.

Le nombre d'octets utilisé pour coder un type VISIBLE STRING correspond directement au nombre d'éléments de la chaîne dans la limite de 126 octets.

5.2.1.4.2.8 Règle de codage des valeurs de type GENERALIZED TIME

Le codage de la valeur est effectué de la même manière que pour une valeur de type VISIBLE STRING de 14 caractères.

Le temps généralisé (spécifié dans l'ISO 8824) est caractérisé par un nombre spécifique de caractères codés sur des octets.

Le temps généralisé est une juxtaposition de caractères dans la séquence suivante:

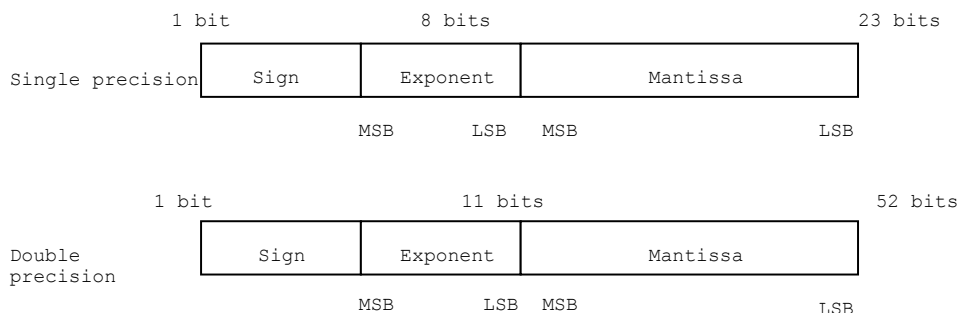
- 4 chiffres représentent l'année
- 2 chiffres représentent le mois
- 2 chiffres représentent le jour
- 2 chiffres représentent l'heure
- 2 chiffres représentent les minutes
- 2 chiffres représentent les secondes

Les premiers octets sont ceux incluant le codage de l'année.

Le temps généralisé exprime directement une heure locale.

5.2.1.4.2.9 Règle de codage des valeurs de type FLOATING POINT

Le codage d'une valeur de type floating point est réalisé sur 4 ou 8 octets selon l'attribut size correspondant à une simple ou à une double précision.



Légende

Anglais	Français
Single precision	Simple précision
Double precision	Double précision
Sign	Signe
Exponent	Exposant
Mantissa	Mantisse

Figure 5 – Codage d'une valeur de type floating point

Les formats de présentation définis dans la CEI 60559 sont indiqués dans la Figure 5.

Les octets de contenu contiennent le signe, l'exposant et la mantisse dans l'ordre précédent à partir du premier jusqu'au dernier octet.

Dans tous les cas le signe est codé en utilisant le bit 8 du premier octet. Il est suivi par l'exposant à partir du bit 7 du premier octet, puis la mantisse commence à partir du bit 7 du second octet pour la simple précision et à partir du bit 4 du second octet pour la double précision.

Le codage de la valeur de la mantisse et de la valeur de l'exposant est décrit dans la CEI 60559.

5.2.1.4.2.10 Règle de codage des valeurs de type BINARY TIME

Le codage d'une valeur de type binary time est réalisé sur un nombre d'octets définis par l'attribut size, et décrit un certain nombre d'unités de temps élémentaires. La valeur de l'unité de temps élémentaire dépend de la taille sélectionnée.

Les octets de contenu contiennent une valeur binaire équivalente à la valeur de temps binaire, et sont composés des bits 8 à 1 sur le premier octet, suivis par les bits 8 à 1 de chaque octet suivant, jusques et y compris les derniers octets de contenu.

NOTE La valeur binaire est obtenue en numérotant les bits des octets de contenu à partir du bit 1 du dernier octet jusqu'au bit 8 du premier octet. À chaque bit mis à 1 est affecté une valeur numérique de 2^{n-1} , où n est la position du bit dans la numérotation.

5.2.1.4.2.11 Règle de codage des valeurs de type BCD

Le codage de type BCD est réalisé sur un octet. La valeur binaire associée à la valeur de type BCD est codée sur les bits 1 à 4 de l'octet et obtenue en allant du bit 1 au bit 4. À chaque bit mis à '1' est attribuée une valeur numérique de 2^{n-1} , où n est la position du bit dans l'octet.

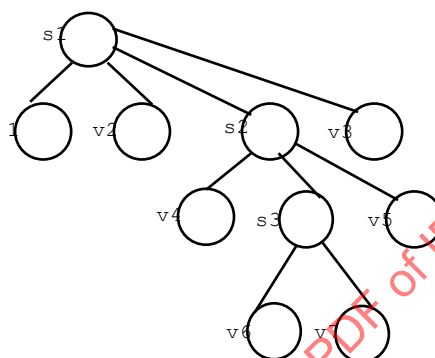
5.2.1.4.2.12 Règle de codage de type structure

Le codage d'un type structure met en jeu les règles de codage de chacun des types simples et une règle de concaténation des types composant cette structure.

Le codage des différents types composant une structure sont juxtaposés dans l'ordre d'apparition au sein de la structure.

La valeur associée au premier champ de structure est codée dans les premiers octets de contenu. La valeur associée au deuxième champ est codée dans les octets suivants, et ainsi de suite, jusqu'au dernier champ de la structure.

Prendre en considération l'arbre type suivant:



Représentation au sein des octets de contenu:

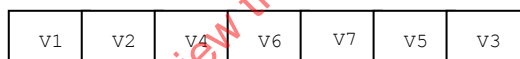


Figure 6 – Codage de type structure

Une structure peut comprendre plusieurs niveaux d'abstraction; dans ce cas, la juxtaposition de codage des valeurs associées aux différents champs est réalisée par un premier parcours approfondi de l'arbre type associé à la structure comme indiqué à la Figure 6.

5.2.1.4.2.13 Codage de type array

Le codage de type array implique la règle de codage d'un type simple et une règle de concaténation des éléments qui composent le tableau.

Les valeurs codées des éléments du tableau sont juxtaposées dans la séquence de déclaration du tableau: ces éléments peuvent être de type simple, structure ou array.

Le premier élément du tableau est contenu dans le premier octet des octets de contenu.

Les types array peuvent éventuellement être définis avec des règles d'optimisation du codage associé.

Ces règles d'optimisation ne font pas obstacle à un accès partiel à un élément du tableau et préservent l'intégrité des données traitées.

5.2.1.4.2.14 Règle de codage compacté de type BOOLEAN array

Le codage normal d'une valeur de type BOOLEAN array utilise un octet pour chaque élément du tableau.

Le codage compacté d'une valeur de type BOOLEAN array est réalisé en utilisant un octet pour un ensemble de huit booléens.

Ainsi, les octets de contenu utilisés pour coder un tableau de type BOOLÉEN compacté sont organisés comme indiqué à la Figure 7.

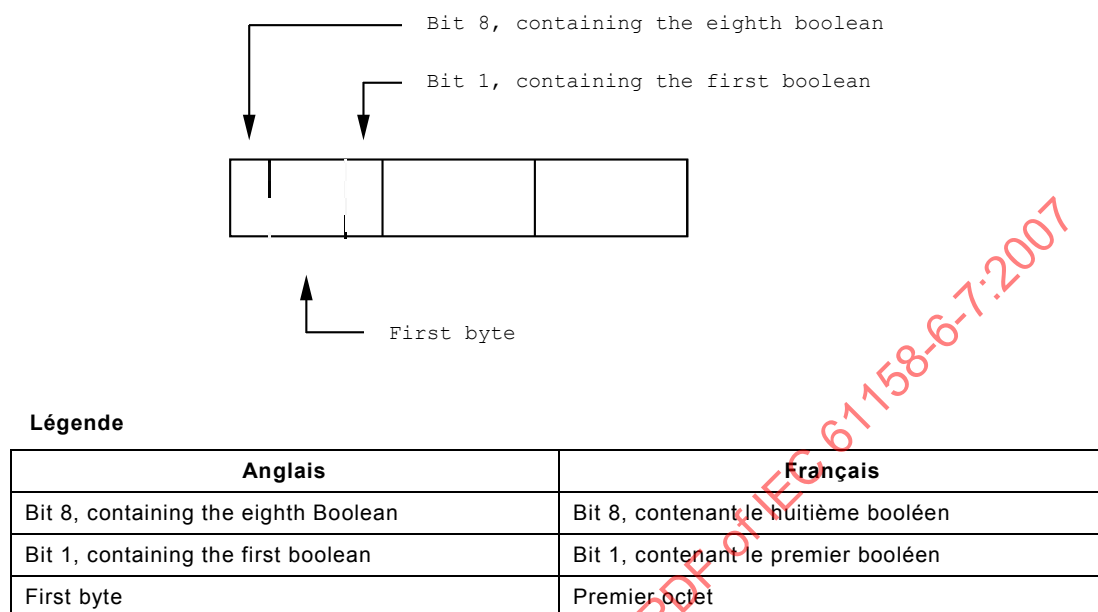


Figure 7 – Codage de type Boolean array

Dans le codage compacté d'un tableau booléen, une valeur de type boolean est 'TRUE' lorsque le bit correspondant est mis à '1'.

Chaque tableau booléen compacté est codé en utilisant un nombre d'octets égal à $E[(e-1)/8] + 1$ où E est la partie entière et e le nombre d'éléments de ce tableau booléen.

5.2.1.4.2.15 Règle de codage compacté de type BCD array

Le codage normal d'une valeur de type BCD array utilise un octet pour chaque élément du tableau.

Le codage compact d'une valeur de type BCD array est effectué à l'aide d'un octet pour une paire de BCD.

Ainsi, au sein des octets de contenu utilisés pour le codage d'un tableau BCD compacté, les bits 8 à 5 du premier octet contiennent le premier BCD du tableau, et les bits 4 à 1 du premier octet contiennent le deuxième BCD du tableau, et ainsi de suite jusqu'au dernier BCD du tableau.

La valeur binaire associée à la valeur du premier BCD d'un octet est obtenue en allant du bit 5 au bit 8: À chaque bit est assignée une valeur numérique de 2^{n-5} , où n est la position du bit au sein de l'octet.

La valeur binaire associée à la valeur du deuxième BCD d'un octet est obtenue en allant du bit 1 au bit 4: à chaque bit est assignée une valeur numérique de 2^{n-1} , où n est la position du bit au sein de l'octet.

5.2.1.4.3 PDU ExplicitVariableValue

La PDU de type Explicit PDU permet de représenter les valeurs des variables pour lesquelles la construction du type est EXPLICITE.

Cette PDU prend en charge les valeurs des variables qui n'ont pas une longueur constante.

ExplicitPDU ::= CHOICE

```
{
  array                [0] IMPLICIT SEQUENCE OF ExplicitPDU,
  structure            [1] IMPLICIT SEQUENCE OF ExplicitPDU,
  boolean              [2] IMPLICIT BOOLEAN,
  bitString            [3] IMPLICIT BITSTRING,
  integer              [4] IMPLICIT INTEGER,
  unsigned             [5] IMPLICIT INTEGER,
  floating             [6] IMPLICIT OCTET STRING,
  octetString          [7] IMPLICIT OCTET STRING,
  visibleString        [8] IMPLICIT OCTET STRING,
  generalizedTime      [9] IMPLICIT OCTET STRING,
  binaryTime          [10] IMPLICIT OCTET STRING,
  bcd                  [1] IMPLICIT INTEGER
}
```

La longueur totale de la PDU ne doit pas dépasser 128 octets.

5.2.1.4.4 PDU VariableDescription

Cette PDU représente la valeur des variables dont le type est K_DVAR. Ce type est utilisé pour les variables de description.

VariableDescriptionPDU ::= SEQUENCE

```
{
  FixedDescription      [0] IMPLICIT Description,
                        -- type prédéfini spécifié à partir d'ici
  variableName          [1] IMPLICIT Symbol,
  resynchronisationReference [2] IMPLICIT ObjectReference OPTIONAL,
  productionPeriod      [3] IMPLICIT Unsigned16 OPTIONAL,
  synchronisationReference [4] IMPLICIT ObjectReference OPTIONAL,
  productionSlot        [5] IMPLICIT Unsigned16 OPTIONAL,
  punctualSynchroReference [6] IMPLICIT ObjectReference OPTIONAL,
  listReference         [7] IMPLICIT ObjectReference OPTIONAL,
  otherDetails          [8] IMPLICIT ANY OPTIONAL,
  vendorInformation     [9] IMPLICIT OCTET STRING OPTIONAL
}
```

La longueur totale de la PDU ne doit pas dépasser 128 octets. Voir Tableau 8.

NOTE Le champ de cette PDU permet le codage des attributs suivants dans les objets de la couche d'application de MPS.

Tableau 8 – Champ d'une VariableDescriptionPDU

Champ	Classe	Attribut
FixedDescription	Variable	Identifier Network Period Class Consistency Variable Significant Status Reference Type Constructor
Variable Name	Variable	A_Name
ResynchronizationReference	Resynchronization in Production	Reference (Synchronization)Variable
ProductionPeriod	Refreshment	Production Variable
SynchronizationReference	Refreshment	Reference (Synchronization)Variable
ProductionSlot	Punctual Refreshment	Production Time Slot
PunctualSynchroReference	Punctual Refreshment	Reference (Synchronization)Variable
ListReference	Variable	Reference Variable List

La description des types de cette PDU utilise le type de Description qui contient la partie non facultative de taille constante de cette PDU.

Description:: = OCTET STRING -- Taille fixée à 7 octets

Sémantique d'octets:

octets 1 + 2: Identifiant de la variable décrite

(Le bit de poids fort de l'identifiant dans le bit 8 du premier octet et le bit de poids faible dans le bit 1 du second octet)

octets 3 + 4: Période de réseau

(Le bit de poids fort de la période de réseau dans le bit 8 du premier octet, en multiples de 1 milliseconde; une valeur 0 indique une variable apériodique)

octet 5: *Bit 8, et 7:* Classe des variables

00: NORMAL

01: SYNCHRONIZATION

10: DESCRIPTION

Toutes les autres combinaisons sont réservées

Bit 6: Sous-classe d'une variable de synchronisation

1: Variable de cohérence

0: Variable de synchronisation mais pas de variable de cohérence

Bits 5 et 4: Statut significatif

00: Statut non significatif

01: Rafraîchissement significatif

10: Rafraîchissement ponctuel significatif

11: Rafraîchissement significatif et rafraîchissement ponctuel

Bits 3 à 1 réservés.

octets 6 + 7: Référence de type

(codé sur 16 bits en utilisant l'identifiant de la variable de description de ce type. Le bit 8 de l'octet 6 étant de poids fort et le bit 1 de l'octet 7 étant de poids faible)

La description de type de cette PDU utilise l'élément non terminal de la grammaire ObjectReference qui se réfère à des variables, des listes et des types au moyen de l'identifiant de la variable de description associée. Ces identifiants sont codés sur deux octets: le bit de poids fort de l'identifiant est codé dans le bit 8 du premier octet et le bit de poids faible dans le bit 1 du second octet.

ObjectReference::= OCTET STRING -- Taille fixée à 2 octets

Les types ANY inclus dans la définition de la PDU, ainsi que les bits réservés au sein de cette PDU sont utilisés pour prendre en charge les compléments de description spécifiés ailleurs.

En outre, des chaînes d'octets facultatives permettent l'insertion d'informations spécifiques au vendeur dans cette PDU.

5.2.1.4.5 PDU AccessDescription

Cette unité PDU prend en charge la valeur des variables dont le type est K_DACCESS. Ce type est utilisé pour les variables de description de l'accès. Voir Tableau 9.

AccessDescriptionPDU ::= SEQUENCE

```
{
  accessName                [0] IMPLICIT Symbol,
  rootReference              [1] IMPLICIT ObjectReference,
  accessPath                 [2] IMPLICIT SEQUENCE OF CHOICE
  {
    structureFieldName       [0] IMPLICIT Symbol,
    arrayElementIndex        [1] IMPLICIT Unsigned8
  } OPTIONAL
  otherDetails               [3] IMPLICIT ANY OPTIONAL,
  vendorInformation          [4] IMPLICIT OCTET STRING OPTIONAL
}
```

NOTE Le champ de cette PDU permet le codage des attributs suivants dans les objets de la couche d'application de MPS.

Tableau 9 – Champ d'une AccessDescriptionPDU

Champ	Classe	Attribut
Access Name	Access	A_Name
Root Reference	Access	Reference Variable
Access Path	Type Constructor	Field Name

Les types ANY incluent des compléments de description spécifiés ailleurs dans la définition de la PDU.

En outre, des chaînes d'octets facultatives permettent l'insertion d'informations spécifiques au vendeur dans cette PDU.

5.2.1.4.6 PDU TypeDescription

Cette PDU prend en charge la valeur des variables dont le type est K_DTYPE. Ce type est utilisé pour les variables de description de type.

TypeDescriptionPDU ::= SEQUENCE

```
{
  typeName                  [0] IMPLICIT Symbol,
  typeDescription           [1] CHOICE
  {
    array                   [0] IMPLICIT SEQUENCE
    {
      compressed            [0] IMPLICIT BOOLEAN DEFAULT FALSE,
      dimension              [1] IMPLICIT Unsigned8,
      elementReference       [2] IMPLICIT ObjectReference
    },
    structure                [1] IMPLICIT SEQUENCE OF SEQUENCE
    {
      fieldName              [0] IMPLICIT Symbol OPTIONAL,
      fieldTypeReference     [1] IMPLICIT ObjectReference
    },
    -- simple
    boolean                  [2] IMPLICIT NULL,
    binaryString              [3] IMPLICIT Unsigned8, -- BITSTRING
    integer                   [4] IMPLICIT Unsigned8, -- INTEGER
    unsigned                  [5] IMPLICIT Unsigned8, -- UNSIGNED
    floatingPoint             [6] IMPLICIT BOOLEAN, -- FLOATING POINT
    octetString               [7] IMPLICIT Unsigned8, -- OCTET STRING
    visibleString             [8] IMPLICIT Unsigned8, -- VISIBLE STRING
    universalTime             [9] IMPLICIT NULL, -- UNIVERSAL TIME
    binaryTime                [10] IMPLICIT Unsigned4, -- BINARY TIME
    bcd                       [11] IMPLICIT NULL, -- BCD
    predefined                [12] IMPLICIT NULL,
    explicit                  [13] IMPLICIT NULL,
    otherDetails              [2] IMPLICIT ANY OPTIONAL,
    vendorInformation         [3] IMPLICIT OCTET STRING OPTIONAL
  }
}
```

-- La longueur totale de la PDU ne doit pas dépasser 128 octets. Voir Tableau 10.

NOTE Le champ de cette PDU permet le codage des attributs suivants dans les objets de la couche d'application de MPS.

Tableau 10 – Champ d'une TypeDescriptionPDU

Champ	Classe	Attribut
TypeName	TypeConstructor	A_Name
Compressed	TypeConstructor	Compressed
Dimension	TypeConstructor	Dimension
ElementReference	TypeConstructor	TypeConstructor
FieldName	TypeConstructor	FieldName
FieldTypeReference	TypeConstructor	TypeConstructor
BinaryString	TypeConstructor	Size
Integer	TypeConstructor	Size
Unsigned	TypeConstructor	Size
FloatingPoint	TypeConstructor	Size
OctetString	TypeConstructor	Size
VisibleString	TypeConstructor	Size
BinaryTime	TypeConstructor	Size

Les types ANY incluent des compléments de description spécifiés ailleurs dans la définition de la PDU.

En outre, des chaînes d'octets facultatives permettent l'insertion d'informations spécifiques aux vendeurs dans cette PDU.

PDU ListDescription

Cette PDU prend en charge la valeur des variables dont le type est K_DLIST. Ce type est utilisé pour les variables de description de liste.

```

ListDescriptionPDU ::= SEQUENCE
{
    listName [0] IMPLICIT Symbol,
    variableReferenceList [1] IMPLICIT ListComponents,
    inconsistencyDetection [2] IMPLICIT Unsigned8 OPTIONAL,
    -- 0 UNDEFINED
    -- 1 PERMANENT
    -- 2 PERMANENT
    -- Les autres valeurs sont réservées
    synchronizationReference [3] IMPLICIT ObjectReference OPTIONAL,
    consistencyVariableList [4] IMPLICIT ListConsistencyVariables OPTIONAL,
    recoveryPeriod [5] IMPLICIT Unsigned16 OPTIONAL,
    otherDetails [6] IMPLICIT ANY OPTIONAL,
    vendorInformation [7] IMPLICIT OCTET STRING OPTIONAL
}

```

-- La longueur totale de la PDU ne doit pas dépasser 128 octets. Voir Tableau 11.

NOTE Le champ de cette PDU permet le codage des attributs suivants dans les objets de la couche application de MPS.

Tableau 11 – Champ d'une ListDescriptionPDU

Champ	Classe	Attribut
ListName	VariableList	A_Name
VariableReferenceList	VariableList	List of Reference ConsumedVariable
InconsistencyDetection	VariableList	InconsistencyDetection
SynchronizationReference	VariableList	Reference (Synchronization)Variable
ConsistencyVariableList	VariableList	List of Reference (Consistency) Variable
RecoveryPeriod	VariableList	RecoveryPeriod

La description du type de cette PDU utilise l'élément non terminal de la grammaire 'listComponents' qui définit le format de représentation des références aux variables qui composent la liste décrite.

ListComponents ::= OCTET STRING

-- La taille de cette chaîne dépend de celle de la liste.

Chaque nom de variable qui comprend cette liste est codé sur deux octets de cette chaîne d'octets, à partir des deux premiers octets jusqu'aux deux derniers octets dans les limites de taille prévues par cette PDU. Les noms des variables de la liste sont représentés sur deux octets de manière unique à l'aide de l'identifiant de la variable de description associée à chaque variable de cette liste. Le bit de poids fort de l'identifiant est codé dans le bit 8 du premier octet et le bit de poids faible dans le bit 1 du second octet.

La description du type de cette PDU utilise l'élément non terminal de la grammaire 'listConsistencyVariables' qui définit le format de représentation des références à des variables de cohérence associées à la liste des variables dans le cas du fonctionnement du mécanisme de cohérence spatiale.

ListConsistencyVariables ::= OCTET STRING -- taille relative au nombre d'entités d'application impliquées dans la cohérence spatiale.

Chaque nom de variable de cohérence qui participe à l'élaboration de la cohérence spatiale de cette liste est codé sur deux octets de cette chaîne d'octets, à partir des deux premiers octets jusqu'aux deux derniers octets dans les limites de taille prévues par cette PDU. Les noms des variables de cohérence sont représentés sur deux octets de manière unique au moyen de l'identifiant de la variable de description associée à chaque variable de cohérence.

Le bit de poids fort de l'identifiant est codé dans le bit 8 du premier octet et le bit de poids faible dans le bit 1 du second octet.

5.2.1.4.7 Extensions de la PDU

Les types ANY incluent des compléments de description spécifiés ailleurs dans la définition de la PDU.

Pour tous les éléments de grammaire qui participent à la définition du module en cours de définition, il est nécessaire de déclarer la fin de ce module.

END -- Fin du Module AL TYPE 7 - -MPS-1

5.2.2 Types PDU AL ASE AR MCS

5.2.2.1 Règles de codage

Toutes les PDU MCS contiennent un nombre entier d'octets. Les octets d'une PDU sont numérotés de 1 à un maximum de 256 pour les plus PDU les plus longues. Ces octets sont transmis par le biais du bus dans l'ordre du premier jusqu'au dernier octet. Chacun des octets d'une PDU possède des bits numérotés de 1 à 8, où le bit 1 est de poids faible et le bit 8 est transmis en premier sur la ligne. Enfin, lorsqu'un certain nombre d'octets consécutifs représente un nombre binaire, l'octet de poids faible contient la valeur la plus significative.

Dans les figures suivantes, les octets les moins significatifs sont toujours placés sur la gauche lorsqu'ils sont représentés horizontalement et au sommet lorsqu'ils sont représentés verticalement. En outre, dans un octet, les bits sont toujours représentés avec le bit 8 sur la gauche et le bit 1 sur la droite.

5.2.2.2 Structure des PDU

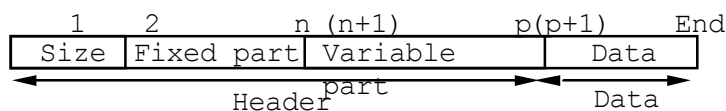
5.2.2.2.1 Définition de premier niveau

En règle générale, toutes les PDU MCS se composent dans l'ordre de:

- l'en-tête contenant:
- la taille de l'en-tête,
- la partie fixe contenant ce qui est obligatoire,

- la partie variable contenant ce qui est facultatif,
- les données qui sont présentes de façon facultative.

La représentation est présentée à la Figure 8



Légende

Anglais	Français
Header	En-tête
Data	Données
End	Fin
Fixed part	Partie fixe
Variable part	Partie variable
Size	Taille

Figure 8 – Représentation d'une PDU MCS

a) Taille:

La taille est égale à la somme des nombres d'octets de la partie fixe et de la partie variable de l'en-tête de la PDU.

La valeur de la taille est codée dans le premier octet de l'en-tête comme un nombre binaire.

La valeur de la taille maximale est de 255 et est codée 1111 1111.

b) Partie fixe

Quel que soit le type de PDU, la partie fixe contient son code suivi du codage d'une liste de paramètres qui sont obligatoires pour la valeur du code.

Les valeurs des codes de PDU associées aux différents types de PDU sont telles que présentées dans le Tableau 12.

Tableau 12 – Codage des différents types de PDU MCS

Valeur de PDU	Signification
1000 0000	demande d'établissement d'association
1100 0000	réponse d'établissement d'association
1001 0000	demande d'interruption d'association
1101 0000	réponse d'interruption d'association
1011 0000	demande de révocation d'association
00X0 0XXX	transfert associé
0110 0000	acquittement du transfert associé
00X1 0XXX	transfert non associé
0111 0XXX	acquittement du transfert non associé

Les bits de certains codes marqués X ont les valeurs indiquées dans les descriptions des types de PDU associés.

NOTE Tous les autres codes sont en attente dans la présente norme.

c) Partie variable

Ce champ est destiné à recevoir les paramètres facultatifs, qui peuvent être présents ou absents selon les classes de conformité des appareils et des exigences de l'utilisateur.

Chaque paramètre de service codé dans la partie variable est structuré comme indiqué dans le Tableau 13.

Tableau 13 – Codage de la partie variable de la PDU

Mapping	Signification
octet 1	Code du paramètre
octet 2	Longueur du paramètre (longueur = m)
octet 2 + 1	Début de la valeur du paramètre
(à)	(suite)
octet 2 + m	Fin de la valeur du paramètre

Chacun des paramètres possède un code de paramètre qui est universel dans l'environnement de MCS, et qui est donc couvert par la présente norme (ceux-ci sont définis pour chaque type de PDU les mettant en œuvre).

La longueur du paramètre est codée dans un octet. La valeur du paramètre est codée dans autant d'octets que spécifié dans la longueur et la sémantique du codage est spécifique à chaque paramètre (la sémantique est définie au niveau de chaque type de PDU le mettant en œuvre).

d) Données

Ce champ est transparent pour MCS, car il contient des données spécifiques à l'un des éléments spécifiques de services d'application projetées sur MCS.

5.2.2.2.2 Demande d'établissement d'association (AARQ)

La structure d'AARQ est telle que présentée au Tableau 14.

Tableau 14 – Structure de la demande d'établissement d'association

Mapping	Signification
octet 1	Taille
octet 2	Code de PDU
octet 3	Point d'accès de l'AEI appelante
octet 4	idem (dito)
octet 5	idem
octet 6	Point d'accès de l'AEI appelée
octet 7	idem
octet 8	idem
octet 9	Classe de conformité
octet 10	Partie variable
à	idem
octet p	idem
octet p+1	Données utilisateur
à	idem
octet de fin	idem

a) Taille

La valeur de la taille minimale est 8 (codé: 0000 1000) lorsque les paramètres relatifs à cette PDU ne sont pas présents, et la valeur de la taille maximale est atteinte lorsque tous les paramètres sont présents.

(Par exemple, la valeur de la taille maximale est de 51 octets.)

b) Partie fixe

Les champs suivants sont présents dans la partie fixe.

- Le code de PDU prend la valeur «1000 0000».

- Le point d'accès de l'AEI appelante, qui correspond à une adresse de liaison (de type ADAEI) est représenté par le bit I/G comme étant le bit de poids le plus fort dans l'octet de poids le plus faible.

Représentation:

```
octet_3:    I / G x x x    x x x x
octet_4:    x x x x x x x x
octet_5:    S / R x x x    x x x x
```

- Le point d'accès de l'AEI appelée, qui correspond à une adresse de liaison (de type ADAEI) est représenté par le bit de poids le plus fort dans l'octet de poids le plus faible.

Représentation:

```
octet_6 :    I / G x x x    x x x x
octet_7 :    x x x x x x x x
octet_8 :    S / R x x x    x x x x
```

NOTE La valeur par défaut utilisée pour ce point d'accès lorsque l'initiateur de l'établissement d'association ne souhaite pas la remplir est la valeur suivante:

```
octet_6 :    1111    1111
octet_7 :    1111    1111
octet_8 :    1111    1110
```

Il s'agit d'une valeur d'adresse de liaison qui se trouve dans l'espace d'adresses de groupe généralisé réservé selon la norme.

- Class of conformity (Classe de conformité)

Représentation:

```
octet_9:    b8 b7 b6 b5 b4 b3 b2 b1
```

Où:

```
b8 = 1, si PV_RE = 2
b7 = 1, si PH_AK = 2
b6 = 1, si PH_RA = 2
b5 = 1, si PH_SR = 2
b4 = 1, si PH_CF = 2
```

Les combinaisons possibles de ces bits doivent tenir compte des contraintes placées sur les classes de conformité

c) Partie variable

Les paramètres suivants doivent être présents dans la partie variable.

- Transfer rate (Taux de transfert):

```
Code du paramètre: 0000 0000
Longueur du paramètre: 4 octets (0000 0100)
Valeur du paramètre:
xxxx xxxx    :    Appelant à Appelé
xxxx xxxx
yyyy yyyy    :    Appelé à Appelant
yyyy yyyy
```

La valeur de ces deux taux est représentée par un nombre binaire égal à l'intervalle de temps entre des messages successifs exprimé en ms.

La valeur 0000 0000 0000 0001 correspond à 1 msg/ms et la valeur 1110 1010 0110 0000 correspond à 1 msg/min.

La valeur 0000 0000 0000 0000 représente conventionnellement un flux minimal garanti nul dans un sens de transfert.

Ce paramètre ne peut être présent que si b4 (PH_CF) du paramètre Class of Conformity est mis à 1.

- Anticipation factor (Facteur d'anticipation):

Code du paramètre: 0000 0001

Longueur du paramètre: 2 octets (0000 0010)

Valeur du paramètre:

xxxx xxxx : Appelant à Appelé

yyyy yyyy : Appelé à Appelant

Le facteur d'anticipation est représenté par un nombre binaire qui permet de différencier 255 niveaux. Le niveau d'anticipation de «i» signifie qu'il est possible d'avoir un maximum de «i» ATAK_SM_RQ actuellement dans l'association.

L'utilisation de la valeur 0 est illégale.

Ce paramètre ne peut être présent que si b7(PH_AK) du paramètre Class of Conformity est mis à 1.

- Application context (Contexte d'application):

Code du paramètre 0000 0010

Longueur du paramètre 3 octets (0000 0011)

Valeur du paramètre

xxxx xxxx : Type d'ASE spécifique

yyyy yyyy : Type de syntaxe abstraite

zzzz zzzz : Type de syntaxe de transfert

La valeur de ces différents types est spécifique à chacun des ASE d'application. Le bit de poids le plus fort de chacun de ces octets indique s'il s'agit d'un type défini dans une norme (b8=1) ou d'un type spécifique à une application (b8=0).

- Identification de l'entité appelante):

Code du paramètre 0000 0011

Longueur du paramètre 6 octets (0000 0110)

Valeur du paramètre

xxxx xxxx : Identifiant d'AEI

yyyy yyyy : Identifiant d'API

zzzz zzzz : Qualificateur d'AE

uuuu uuuu : Titre d'AP

uuuu uuuu

uuuu uuuu

Le premier octet de la valeur désigne l'identifiant d'AEI au moyen d'un nombre binaire qui permet de différencier 255 AEI pour une API dans le cadre d'une AE donnée. La valeur 0000 0000 est la valeur par défaut utilisée pour ne pas adopter ce paramètre.

Le deuxième octet désigne l'identifiant d'API au moyen d'un nombre binaire qui permet de différencier 255 API pour un AP donné. La valeur 0000 0000 est la valeur par défaut utilisée pour ne pas adopter ce paramètre.

Le troisième octet désigne le qualificateur d'AE au moyen d'un nombre binaire qui permet de différencier 255 AE pour un AP donné. La valeur 0000 0000 est la valeur par défaut utilisée pour ne pas adopter ce paramètre.

Les derniers octets libres désignent le titre d'AP au moyen d'un nombre binaire. Ces trois octets sont mis à 0000 0000 si ce paramètre n'est pas adopté.

- Called entity identification (Identification de l'entité appelée):

Code du paramètre 0000 0100
 Longueur du paramètre 6 octets (0000 0110)
 Valeur du paramètre
 xxxx xxxx : Identifiant d'AEI
 yyyy yyyy : Identifiant d'API
 zzzz zzzz : Qualificateur d'AE
 uuuu uuuu : Titre d'AP
 uuuu uuuu
 uuuu uuuu

Le premier octet de la valeur désigne l'identifiant d'AEI au moyen d'un nombre binaire qui permet de différencier 255 AEI pour une API dans le cadre d'une AE donnée. La valeur 0000 0000 est la valeur par défaut utilisée pour ne pas adopter ce paramètre.

Le deuxième octet désigne l'identifiant d'API au moyen d'un nombre binaire qui permet de différencier 255 API pour un AP donné. La valeur 0000 0000 est la valeur par défaut utilisée pour ne pas adopter ce paramètre.

Le troisième octet désigne le qualificateur d'AE au moyen d'un nombre binaire qui permet de différencier 255 AE pour un AP donné. La valeur 0000 0000 est la valeur par défaut utilisée pour ne pas adopter ce paramètre.

Les trois derniers octets désignent le titre d'AP au moyen d'un nombre binaire. Ces trois octets sont mis à 0000 0000 si ce paramètre n'est pas adopté.

- SDU size (Taille de SDU):

Code du paramètre 0000 0101
 Longueur du paramètre 2 octets (0000 0010)
 Valeur du paramètre
 xxxx xxxx : Appelant à Appelé
 yyyy yyyy : Appelé à Appelant

Le paramètre DSU size est représenté par un nombre binaire qui fournit une taille de SDU maximale de 256 DLPDU, soit 64 octets pour une taille de DLPDU maximale de 256 octets.

La valeur 0 de ce paramètre est illégale.

Ce paramètre ne peut être présent que si b5(PH_SR) du paramètre Class of conformity est mis à 1.

- PDU size (Taille de PDU):

Code du paramètre 0000 0110
 Length: 1 octet (0000 0001)
 Valeur du paramètre
 xxxx xxxx : Taille de SDU

La taille de la PDU est représentée par un nombre binaire permettant une taille de PDU maximale de 256 octets.

Ce paramètre peut être présent quelle que soit la valeur de la classe de conformité.

- Number of retries (Nombre de nouvelles tentatives):

Code du paramètre 0000 0111

Length: 1 octet (0000 0001)

Valeur du paramètre 0000 xxxx

Le nombre de nouvelles tentatives est représenté par un nombre binaire permettant un maximum de 16 tentatives codées en bits xxxx.

La valeur 0 de ce paramètre est illégale.

Ce paramètre ne peut être présent que si b6(PH_RA) du paramètre Class of Conformity est mis à 1.

- “Cyclic flow” option (option de «flux cyclique»):

Code du paramètre 0000 1000

Length: 0 octet

d) Données

Les données transportées dans cette PDU sont codées de manière interprétable dans le contexte de l'application.

5.2.2.2.3 Réponse d'établissement d'association (AARP)

La structure d'AARP est présentée au Tableau 15.

Tableau 15 – Structure d'une réponse d'établissement associé

Mapping	Signification
octet 1	Taille
octet 2	Code de PDU
octet 3	Point d'accès de l'AEI appelante
octet 4	idem
octet 5	idem
octet 6	Point d'accès de l'AEI appelée
octet 7	idem
octet 8	idem
octet 9	Classe de conformité
octet 10	Résultat
octet 11	Partie variable
à	idem
octet p	idem
octet p+1	Données utilisateur
à	idem
octet de fin	idem

a) Taille

La valeur de la taille minimale est de 9 octets (codée 0000 1001), lorsque les paramètres relatifs à cette PDU ne sont pas présents et la valeur de taille maximale est atteinte lorsque tous les paramètres sont présents.

b) Partie fixe

Les champs suivants sont présents dans cette partie.

- Le code de PDU prend la valeur «1100 0000».

- Le point d'accès de l'AEI appelante, qui correspond à une adresse de liaison (de type ADAEI) est représenté par le bit I/G comme étant le bit de poids le plus fort dans l'octet de poids le plus faible.

Représentation:

```

octet_3:      I / G x x x      x x x x
octet_4:      x x x x      x x x x
octet_5:      S / R x x x      x x x x
    
```

- Le point d'accès de l'AEI appelée, qui correspond à une adresse de liaison (de type ADAEI) est représenté par le bit de poids le plus fort dans l'octet de poids le plus faible.

Représentation:

```

octet_6:      I / G x x x      x x x x
octet_7:      x x x x      x x x x
octet_8:      S / R x x x      x x x x
    
```

NOTE La valeur par défaut utilisée pour ce point d'accès lorsque l'initiateur de l'établissement d'association ne souhaite pas la remplir est la valeur suivante:

```

octet_6: 1111      1111
octet_7: 1111      1111
octet_8: 1111      1110
    
```

Il s'agit d'une valeur d'adresse de liaison qui se trouve dans l'espace d'adresses de groupe généralisé réservé selon la norme.

- Class of conformity (Classe de conformité)

Représentation:

```

octet_9:      b8 b7 b6 b5 b4 b3 b2 b1
Où:
b8 = 1, si PV_RE = 2
b7 = 1, si PH_AK = 1
b6 = 1, si PH_RA = 2
b5 = 1, si PH_SR = 2
b4 = 1, si PH_CF = 2
    
```

Les combinaisons possibles de ces bits doivent tenir compte des contraintes placées sur les classes de conformité.

- Le résultat qui suit une demande d'établissement d'association est codé dans un octet et indique si la réponse est positive ou négative, dans ce dernier cas, il fournit des informations sur la nature et l'emplacement de du défaut.

Les valeurs de l'octet de niveau 10 contiennent les valeurs suivantes dans les cas suivants:

```

0000 0000: (+) Positive
0000 0001: (-) Fournisseur de MCS (général)
0000 0010: (-) Utilisateur répondeur
0000 0011: (-) lié à «Cyclic rate»
0000 0100: (-) lié à «Transfer rate»
0000 0101: (-) lié à «Context name»
0000 0110: (-) lié à «Called entity identification»
0000 0111: (-) lié à «Calling entity identification»
0000 1000: (-) lié à «Service size»
xxxx xxxx: Double
    
```

c) Partie variable

Les paramètres suivants peuvent être présents dans la partie variable.

- Transfer rate (Taux de transfert):

Le codage et la sémantique de ce paramètre sont équivalents à ceux du paramètre portant le même nom dans la demande d'établissement d'association.

- Anticipation factor (Facteur d'anticipation):

Le codage et la sémantique de ce paramètre sont équivalents à ceux du paramètre portant le même nom dans la PDU de demande d'établissement d'association.

- Application context (Contexte d'application):

Le codage et la sémantique de ce paramètre sont équivalents à ceux du paramètre portant le même nom dans la PDU de demande d'établissement d'association (voir 10.2.3.2.2.c).

- Called entity identification (Identification de l'entité appelée):

Le codage et la sémantique de ce paramètre sont équivalents à ceux du paramètre portant le même nom dans la PDU de demande d'établissement d'association.

- SDU size (taille de SDU):

Le codage et la sémantique de ce paramètre sont équivalents à ceux du paramètre portant le même nom dans la PDU de demande d'établissement d'association (voir 10.2.3.2.2.c).

- PDU size (taille de PDU):

Le codage et la sémantique de ce paramètre sont équivalents à ceux du paramètre portant le même nom dans la PDU de demande d'établissement d'association, (voir 10.2.3.2.2.c).

- Number of retries (Nombre de nouvelles tentatives):

Le codage et la sémantique de ce paramètre sont équivalents à ceux du paramètre portant le même nom dans la PDU de demande d'établissement d'association (voir 10.2.3.2.2.c).

d) Données

Les données transportées dans cette PDU sont codées de manière interprétable dans le contexte de l'application.

5.2.2.2.4 Demande de rupture d'association (RERQ)

La structure de RERQ est indiquée dans le Tableau 16.

Tableau 16 – Structure d'une demande de rupture d'association

Mapping	Signification
octet 1	Taille
octet 2	Code de PDU
octet 3	Données utilisateur
à	idem
octet de fin	idem

a) Taille

La valeur de la taille est de 1 octet (codé 0000 0001)

b) Partie fixe

Le seul champ contenant la partie fixe est PDU code (code de PDU) qui prend la valeur «1001 0000».

c) Partie variable

Il n'existe pas de partie variable dans cette PDU.

d) Données

Les données transportées dans cette PDU sont codées de manière interprétable dans le contexte de l'application.

5.2.2.2.5 Réponse d'interruption d'association (RERP)

La structure de RERP est indiquée dans le Tableau 17.

Tableau 17 – Structure d'une réponse d'interruption d'association

Mapping	Signification
octet 1	Taille
octet 2	Code de PDU
octet 3	Résultat
octet 4	Données utilisateur
à	idem
octet de fin	idem

a) Taille

La valeur de la taille est de 2 octets (codée: 0000 0010).

b) Partie fixe

La partie fixe de l'en-tête contient les deux champs suivants:

- PDU code (code de PDU): "1101 0000"
- Result (Résultat) qui est codé dans un octet, indique dans un premier temps si le résultat est positif ou négatif et, dans un deuxième temps, fournit des informations sur le défaut, à définir ultérieurement.

0000 0000: (+) Positive
 0000 0001: (-) Négative
 xxxx xxxx: Double (Défauts)

c) Partie variable

Cette PDU ne comporte aucune partie variable.

d) Données

Les données transportées dans cette PDU sont codées de manière interprétable dans le contexte de l'application.

5.2.2.2.6 Demande de révocation d'association (ABRQ)

La structure d'ABRQ est indiquée dans le Tableau 18.

Tableau 18 – Structure d'une demande de révocation d'association

Mapping	Signification
octet 1	Taille
octet 2	Code de PDU
octet 3	Origine de l'invocation
octet 4	Données utilisateur
à	idem
octet de fin	idem

a) Taille

La valeur de la taille est de 2 octets (codée: 0000 0010).

b) Partie fixe

Les deux champs suivants forment la partie fixe:

- PDU code (code de PDU) prend la valeur «1011 0000».
- Invocation origin (origine de l'invocation) est codé sur un octet et prend deux valeurs:

0000 0000: Utilisateur de MCS
 1111 1111: Fournisseur de MCS
 xxxx xxxx: Autres codes réservés

c) Partie variable

Cette PDU ne comporte aucune partie variable.

d) Données

Les données transportées dans cette PDU sont codées de manière interprétable dans le contexte de l'application.

5.2.2.2.7 Demande de transfert associé (ATRQ)

La structure d'ATRQ est indiquée dans le Tableau 19.

Tableau 19 – Structure d'une demande de transfert associé

Mapping	Signification
octet 1	Taille
octet 2	Code de PDU
octet 3	Numéro de séquence
octet 4	idem
octet 5	Données utilisateur
à	idem
octet de fin	idem

a) Taille

La valeur de la taille est de 3 octets (codée: 0000 0011).

b) Partie fixe

Les deux champs suivants constituent la partie fixe de l'en-tête:

- PDU code (code de PDU) prend les valeurs:

0000 0000: pour les transferts non acquittés
 0010 0XXX: pour les transferts acquittés

où XXX est utilisé pour coder la nature des paquets dans le cas de la segmentation.

- Packet nature (nature des paquets) est codé comme suit:
 - 000: NIL (pas de segmentation)
 - 100: STT (premier paquet)
 - 101: MID (paquet intermédiaire)
 - 110: END (dernier paquet)
 - 111: ALL (premier et dernier paquet).
 - Sequence number (numéro de séquence) est représenté dans deux octets au moyen d'un nombre binaire. Cela permet de différencier les 65536 transferts de données. Ce nombre n'est pas significatif dans le cas des transferts non acquittés.
- c) Partie variable
Cette PDU ne comporte aucune partie variable.
- d) Données
Les données transportées dans cette PDU sont codées de manière interprétable dans le contexte de l'application spécifique à l'association.

5.2.2.2.8 Acquittement de transfert associé (AKAT)

La structure d'AKAT est indiquée dans le Tableau 20.

Tableau 20 – Structure d'un acquittement de transfert associé

Mapping	Signification
octet 1	Taille
octet 2	Code de PDU
octet 3	Numéro de séquence
octet 4	idem

- a) Taille
La valeur de la taille est de 3 octets (codée: 0000 0011).
- b) Partie fixe
Le champ suivant est contenu dans la partie fixe:
- PDU number (numéro de PDU) prend la valeur «0110 0000».
 - Sequence number (numéro de séquence) est attribué par l'entité de numérotation.
- c) Partie variable
Cette PDU ne comporte aucune partie variable.
- d) Données
Cette PDU ne comporte aucune donnée utilisateur.

5.2.2.2.9 Demande de transfert non associé (NTRQ)

La structure de NTRQ est indiquée dans le Tableau 21.

Tableau 21 – Structure d'une demande de transfert non associé

Mapping	Signification
octet 1	Taille
octet 2	Code de PDU
octet 3	Identification d'invocation
octet 4	idem
octet 5	Partie variable
à	idem
octet p	idem
octet p+1	Données utilisateur
à	idem
octet de fin	idem

a) Taille

La valeur de la taille minimale est de 3 octets (codée: 0000 0011) lorsque les paramètres relatifs au paramètre de la PDU ne sont pas présents et la valeur maximale est atteinte lorsque tous les paramètres sont présents.

b) Partie fixe

Les champs suivants sont présents dans la partie fixe:

- PDU code (code de PDU) prend les valeurs:

0001 0xxx: pour les transferts non acquittés

0011 0xxx: pour les transferts acquittés

où la valeur de xxx est utilisée pour coder la priorité de service.

- Service priority (priorité de service), codé sur 3 bits permet d'établir 8 niveaux de priorité.
- Invocation identification (identification d'invocation) est représentée dans deux octets par un nombre binaire. Cela permet de différencier 65536 transferts de données simultanés actuels.

c) Partie variable

Les paramètres suivants peuvent être présents dans la partie variable:

- Application context (Contexte d'application)

Le codage et la sémantique de ce paramètre sont équivalents à ceux du paramètre portant le même nom dans la PDU de demande d'établissement d'association (voir 5.2.2.2.2).

- Calling entity identification (identification de l'entité appelante)

Le codage et la sémantique de ce paramètre sont équivalents à ceux du paramètre portant le même nom dans la PDU de demande d'établissement d'association (voir 5.2.2.2.2).

- Called entity identification (identification de l'entité appelée)

Le codage et la sémantique de ce paramètre sont équivalents à ceux du paramètre portant le même nom dans la PDU de demande d'établissement d'association (voir 5.2.2.2.2).

d) Données

Les données transportées dans cette PDU sont codées de manière interprétable dans le contexte de l'application.

5.2.2.2.10 Acquittement de transfert non associé (AKNT)

La structure d'AKNT est indiquée dans le Tableau 22.

Tableau 22 – Structure d'un acquittement de transfert non associé

Mapping	Signification
octet 1	Taille
octet 2	Code de PDU
octet 3	Identification d'invocation
octet 4	ditto

a) Taille

La valeur de la taille est de 3 octets (codée: 0000 0011).

b) Partie fixe

Le champ suivant est contenu dans la partie fixe:

- PDU number (numéro de PDU) prend la valeur «0111 0XXX».

Les 3 bits XXX sont réservés pour le codage d'un défaut qui peut apparaître au niveau de l'accepteur.

Le codage du défaut est comme suit:

000: AKNT+, aucun défaut

≠000: AKNT-, défaut

où:

001: ligne de défaut non spécialisé

010 à 111: défauts, dont la spécialisation est réservée à cette norme.

NOTE L'acquittement n'accorde pas la priorité à la demande de transfert qui a donné lieu à l'acquittement d'autant qu'il est implicite que tout demandeur réservera les ressources pour recevoir l'acquittement qu'il attend.

- Invocation identification (identification d'invocation) est représenté par deux octets avec un nombre binaire. Cela permet de discriminer 65536 transferts de données simultanés actuels.

c) Partie variable

Cette PDU ne comporte aucune partie variable.

d) Données

Cette PDU ne comporte aucune donnée utilisateur.

5.2.3 Type PDU FAL ASE SUB MMS

5.2.3.1 PDU-SUB-MMS

5.2.3.1.1 Définition de premier niveau

Cette section décrit les PDU utilisées dans le protocole SUB-MMS.

SUB-MMS-General-module-1 {iso standard 9605
 part-protocol (2)
 sub-mms-general-module-version1 (5) }

DEFINITION:: = BEGIN

EXPORT SUB-MMSPdu;

-- Par cette affirmation, nous entendons que toute production soumise a SUB-MMSPdu est exportable.--

```
SUB-MMSPdu ::= CHOICE {
  confirmed-reqPDU      [0] IMPLICIT ConfirmedReqPDU,
  confirmed-respPDU     [1] IMPLICIT ConfirmedRespPDU,
  confirmed-errorPDU    [2] IMPLICIT ConfirmedErrorPDU,
  unconfirmedPDU       [3] IMPLICIT UnconfirmedPDU,
  rejectPDU            [4] IMPLICIT RejectPDU,
  initiate-reqPDU      [5] IMPLICIT InitiateReqPDU,
  initiate-respPDU     [6] IMPLICIT InitiateRespPDU,
  initiate-errorPDU    [7] IMPLICIT InitiateErrorPDU,
  conclude-reqPDU      [8] IMPLICIT ConcludeReqPDU,
  conclude-respPDU     [9] IMPLICIT ConcludeRespPDU,
  conclude-errorPDU    [10] IMPLICIT ConcludeErrorPDU
}
```

Le protocole SUB-MMSPdu définit 11 types de PDU

Les Normes d'accompagnement peuvent allonger cette liste.

5.2.3.1.2 Définition de confirmed-req-PDU

```
ConfirmedReqPDU:: = SEQUENCE {
  invoke-ID          UNSIGNED-32,
  confirmed-service-req ConfirmedServiceReq}.

```

La confirmed-req-pdu contient deux paramètres:

- invoke-ID a la même signification que celle donnée dans l'ISO/CEI 9506-2.
- ConfirmedServiceReq est défini ci-dessous.

5.2.3.1.3 Définition d'unconfirmed-PDU

```
UnconfirmedPDU:: = SEQUENCE {
  invoke-ID          UNSIGNED-32,
  unconfirmed-service UnconfirmedService}.

```

L'unconfirmed-pdu contient deux paramètres:

- invoke-ID a la même signification que celle donnée dans l'ISO/CEI 9506-2.

UnconfirmedService est défini ci-dessous.

5.2.3.1.4 Définition de confirmed-resp-pdu

```
ConfirmedRespPDU:: = SEQUENCE {
  invoke-ID          UNSIGNED-32,
  confirmed-service-resp ConfirmedServiceResp}.

```

La confirmed RespPDU contient deux paramètres:

L'invoke-ID a la même signification que celle donnée dans l'ISO/CEI 9506-2.

Le ConfirmedServiceResp est défini ci-dessous.

5.2.3.1.5 Définition de confirmed-error-pdu

```
ConfirmedErrorPDU:: = SEQUENCE {
  invoke-ID          UNSIGNED-32,
  service-error      ErrorType}.

```

La ConfirmedErrorPDU contient deux paramètres:

- invoke-ID a la même signification que celle donnée dans l'ISO/CEI 9506-2.

Error est défini ci-dessous.

5.2.3.1.6 Définition d'«objet» et de «service»

5.2.3.1.6.1 Définition d'une classe d'objets

Une classe d'objets est une expression qualificative attribuée à un type d'ensemble d'objets pour exprimer le fait que la même action appliquée à ces objets donne toujours le même résultat. Nous attribuons un nom à chaque classe d'objets. Par conséquent, une classe d'objets doit être identifiée par son nom. Les normes d'accompagnement (CS) peuvent élargir cette classe d'objets. Voir Tableau 23.

Tableau 23 – Définitions des classes d'objets

Nom de classe	Type d'objet	Création
vmd	visible part of the server	I/T
dom	domain	I/T/E
pi	executable program	I/T/E
var	variable	I/T
var-list	variables-list	I/T/E
event	event	I/T
association	connection	I/T/E
OD-header	OD-header	I/T
data type	type descriptor	I/T
LÉGENDE: I = Création implicite. T = Création faisant suite à un chargement à distance. E = Création Explicite par le Service.		

5.2.3.1.6.2 Définition d'un service

Un type d'action appliqué à un objet appartenant à une classe donnée est appelé service. Nous attribuons un nom à chaque service. Par conséquent, un service sera identifié par son nom. Les normes d'accompagnement peuvent allonger cette liste de services. Voir Tableau 24.

Tableau 24 – Définition des Services de Sub MMS

Nom de Service		Classe d'Objets	Note
services confirmés 1:	Status	vmd	note 1
	Identify	vmd	note 1
	Get Name List	vmd	note 1
	Delete-domain	dom	note 1
	Initiate-download-sequence	dom	note 1
	Download-segment	dom	note 1
	Terminate-download-sequence	dom	note 1
	Initiate-upload-sequence	dom	note 1
	Upload-segment	dom	note 1
	Terminate-upload-sequence	dom	note 1
	Get Domain Attributes	dom	note 1
	Create-program-invocation	pi	note 1
	Delete-program-invocation	pi	note 1
	Start	pi	note 1
	Stop	pi	note 1
	Resume	pi	note 1
	Reset	pi	note 1
	Kill	pi	note 1
	Get Program Invocation Attributes	pi	note 1
	Read	var/var-list	note 1
	Write	var/var-list	note 1
	Get Variable Access Attributes	var	note 1
	Define-variable-list	var-list	note 1
	Delete-variable-list	var-list	note 1
	Get Variable List Attributes	var-list	note 1
	Get-alarm-summary	event	note 1
	Acknowledge-event-notification	event	note 1
	Alter-event-condition-monitoring	event	note 1
	Get Event Condition Attributes	event	note 1
	Initiate	association	note 2
	Conclude	association	note 1
	Generic-Init-Download	OD	note 1
	Generic-Download	OD	note 1
	Generic-terminate-Download	OD	note 1
	Get OD Header/Data Type Attributes	OD	note 1
services non confirmés 2:	Unsolicited-Status	Vmd	note 1
	Information-Report	Var	note 1
	Event-Notification	Event	note 1
	Abort	association	note 2
	Reject	PDU	note 3
NOTE 1 service négociable par le biais du service Initiate. NOTE 2 service obligatoire si les équipements prennent en charge la connexion négociable. NOTE 3 service obligatoire pour tous les équipements prenant en charge les indications des services confirmés (rejet des demandes de services confirmés).			

5.2.3.1.7 Définition de confirmed-service-req

La structure d'une demande de service comprend:

- 1) l'étiquette du service ([X] = code du service),
- 2) les paramètres du service (appelés «req-serv»),
- 3) les extensions de paramètres assurées par les CS (appelées «req-detail»).

Par souci de commodité, les paramètres 2 et 3 sont décrits séparément.

NOTE Les demandes et les réponses de services confirmés sont identifiés par le même code de service. Les demandes de services non confirmés sont identifiées par des codes de services différents de ceux des services confirmés.

ConfirmedServiceReq:: =

--Services DOMAIN--

initiate-download-sequence
req-serv
req-detail
download-segment
req-serv
req-detail
terminate-download-sequence
req-serv
req-detail
initiate-upload-sequence
req-serv
req-detail
upload-segment
req-serv
req-detail
terminate-upload-sequence
req-serv
req-detail
delete-domain
req-serv
req-detail

--Services VMD

status
req-serv
req-detail
identify
req-serv
req-detail

--Services PROGRAM INVOCATION--

create-program-invocation
req-serv
req-detail
delete-program-invocation
req-serv
req-detail
start
req-serv
req-detail
stop
req-serv
req-detail

resume
req-serv
req-detail

reset
req-serv
req-detail

kill
req-serv
req-detail

--Services VARIABLE /VARIABLE-LIST--

read
req-serv
req-detail

write
req-serv
req-detail

define-variable-list
req-serv
req-detail

delete-variable-list
req-serv
req-detail

--Services EVENT--

get-alarm-summary
req-serv
req-detail

acknowledge event notification
req-serv
req-detail

alter-event-cond-monitor
req-serv
req-detail

CHOICE {

- [6] IMPLICIT SEQUENCE {
[0] IMPLICIT InitiateDownloadSequenceRequest,
[1] IMPLICIT CSInitiateDownloadSequenceRequest},
- [7] IMPLICIT SEQUENCE {
[0] DownloadSegmentRequest,
[1] IMPLICIT CSDownloadSegmentRequest},
- [8] IMPLICIT SEQUENCE {
[0] IMPLICIT TerminateDownloadSequenceRequest,
[1] IMPLICIT CSTerminateDownloadSequenceRequest},
- [9] IMPLICIT SEQUENCE {
[0] InitiateUploadSequenceRequest,
[1] IMPLICIT CSInitiateUploadSequenceRequest},
- [10] IMPLICITSEQUENCE {
[0] UploadSegmentRequest,
[1] IMPLICIT CSUploadSegmentRequest},
- [11] IMPLICIT SEQUENCE {
[0] TerminateUploadSequenceRequest,
[1] IMPLICIT CSTerminateUploadSequenceRequest},
- [12] IMPLICITSEQUENCE {
[0] DeleteDomainRequest,
[1] IMPLICIT CSDeleteDomainRequest},
- [15] IMPLICITSEQUENCE {
[0] IMPLICIT StatusRequest,
[1] IMPLICIT CSStatusRequest},
- [16] IMPLICIT SEQUENCE {
[0] IMPLICIT IdentifyRequest,
[1] IMPLICIT CSIdentifyRequest},
- [20] IMPLICITSEQUENCE {
[0] IMPLICIT CreateProgramInvocationRequest,
[1] IMPLICIT CSCreateProgramInvocationRequest},
- [21] IMPLICITSEQUENCE {
[0] DeleteProgramInvocationRequest,
[1] IMPLICIT CSDeleteProgramInvocationRequest},
- [22] IMPLICITSEQUENCE {
[0] IMPLICIT StartRequest,
[1] IMPLICIT CSStartRequest},
- [23] IMPLICITSEQUENCE {
[0] StopRequest,
[1] IMPLICIT CSStopRequest},
- [24] IMPLICITSEQUENCE {
[0] IMPLICIT ResumeRequest,
[1] IMPLICIT CSResumeRequest},
- [25] IMPLICITSEQUENCE {
[0] ResetRequest,
[1] IMPLICIT CSResetRequest},
- [26] IMPLICITSEQUENCE {
[0] KillRequest,
[1] IMPLICIT CSKillRequest},
- [30] IMPLICITSEQUENCE {
[0] ReadRequest,
[1] IMPLICIT CSReadRequest},
- [31] IMPLICITSEQUENCE {
[0] IMPLICIT WriteRequest,
[1] IMPLICIT CSWriteRequest},
- [35] IMPLICITSEQUENCE {
[0] IMPLICIT DefineVariableListRequest,
[1] IMPLICIT CSDefineVariableListRequest},
- [36] IMPLICITSEQUENCE {
[0] DeleteVariableListRequest,
[1] IMPLICIT CSDeleteVariableListRequest},
- [39] IMPLICITSEQUENCE {
[0] GetAlarmSummaryRequest,
[1] IMPLICIT CSGetAlarmSummaryRequest},
- [40] IMPLICITSEQUENCE {
[0] IMPLICIT AcknowledgeEventNotificationRequest,
[1] IMPLICIT CSAcknowledgeEventNotificationRequest},
- [41] IMPLICITSEQUENCE {
[0] IMPLICIT AlterEventConditionMonitoringRequest,
[1] IMPLICIT CSAAlterEventConditionMonitoringRequest},

```

--Services Get Attributes--
  get-name-list          [45] IMPLICITSEQUENCE {
    req-serv              [0] IMPLICIT GetNameListRequest,
    req-detail            [1] IMPLICIT CSGetNameListRequest},
  get-domain-attributes  [46] IMPLICITSEQUENCE {
    req-serv              [0] GetDomainAttributesRequest,
    req-detail            [1] IMPLICIT CSGetDomainAttributesRequest},
  get-program-invocation-attributes [47] IMPLICITSEQUENCE {
    req-serv              [0] GetProgramInvocationAttributesRequest,
    req-detail            [1] IMPLICIT CSGetProgramInvocationAttributesRequest},
  get-variable-access-attributes [48] IMPLICITSEQUENCE {
    req-serv              [0] GetVariableAccessAttributesRequest,
    req-detail            [1] IMPLICIT CSGetVariablesAccessAttributesRequest},
  get-variable-list-attributes [49] IMPLICITSEQUENCE {
    req-serv              [0] GetVariableListAttributesRequest,
    req-detail            [1] IMPLICIT CSGetVariableListAttributesRequest},
  get-event-condition-attributes [50] IMPLICITSEQUENCE {
    req-serv              [0] GetEventConditionAttributesRequest,
    req-detail            [1] IMPLICIT CSGetEventConditionAttributesRequest},
--Services OD--
  generic-init-download  [55] IMPLICITSEQUENCE {
    req-serv              [0] IMPLICIT GenericInitDownloadRequest,
    req-detail            [1] IMPLICIT CSGenericInitDownloadRequest},
  generic-download       [56] IMPLICITSEQUENCE {
    req-serv              [0] IMPLICIT GenericDownloadRequest,
    req-detail            [1] IMPLICIT CSGenericDownloadRequest},
  generic-terminate-download [57] IMPLICITSEQUENCE {
    req-serv              [0] IMPLICIT GenericTerminateDownloadRequest,
    req-detail            [1] IMPLICIT CSGenericTerminateDownloadRequest},
  get-OD-header/datatype-attributes [58] IMPLICITSEQUENCE {
    req-serv              [0] IMPLICIT GetODHeaderDataTypeRequest,
    req-detail            [1] IMPLICIT CSGetODHeaderDataTypeRequest},
}

```

La ConfirmedServiceReq comprend:

- 1) [X] qui identifie le service,
- 2) req-serv fait référence à la production correspondant au service sélectionné,
- 3) req-detail fait référence à la production des Normes d'Accompagnement (CS) correspondant au service sélectionné.

5.2.3.1.8 Définition d'unconfirmed-service

Les services non confirmés sont transmis spontanément. La raison de leur émission n'est pas définie par le présent document (initiative locale).

Le choix de la (des) voie(s) par laquelle (lesquelles) un service non confirmé est transmis constitue également une initiative locale. Par conséquent, une notification peut avoir lieu en point à point et/ou en multipoint et/ou en diffusion.

La structure du service inclut:

- 1) l'étiquette du service ([X] = code du service),
- 2) les paramètres du service (appelés «req-serv»),
- 3) les extensions des paramètres par les Normes d'Accompagnement (appelées «req-detail»).

Par souci de commodité, les paramètres 3 et 4 sont décrits séparément.

NOTE Les demandes de services non confirmés sont identifiées par des codes de services différents de ceux des services confirmés.

```

UnconfirmedService ::= CHOICE {
--Service VMD--
    unsolicited-status          [3] IMPLICIT SEQUENCE {
        req-serv                [0] IMPLICIT UnsolicitedStatus,
        req-detail              [1] IMPLICIT CSUnsolicitedStatus},
--Service EVENT:
    event-notification          [4] IMPLICIT SEQUENCE {
        req-serv                [0] IMPLICIT EventNotification,
        req-detail              [1] IMPLICIT CSEventNotification},
--Service VARIABLE--
    information-report          [5] IMPLICIT SEQUENCE {
        req-serv                [0] IMPLICIT InformationReport,
        req-detail              [1] IMPLICIT CSInformationReport},
}

```

L'unconfirmedService comprend:

- 1) [X] qui identifie le service,
- 2) La référence req-serv de la production correspondant au service sélectionné,
- 3) req-detail qui fait référence à la production des Normes d'Accompagnement (CS) correspondant au service sélectionné.

5.2.3.1.9 Définition de confirmed-service-resp

La structure du service comprend:

- 1) l'étiquette du service ([X] = code du service),
- 2) les paramètres du service (appelés «rsp-serv»),
- 3) les extensions des paramètres effectuées par les Normes d'Accompagnement (appelées «rsp-detail»).

Par souci de commodité, les paramètres 2 et 3 sont décrits autre part.

NOTE Les demandes et les réponses de services confirmés sont identifiées par le même code de service. Les demandes de services non confirmés sont identifiées par des codes de services différents de ceux des services confirmés.

```

ConfirmedServiceResp ::= CHOICE {
--Services DOMAIN--
    initiate-download-sequence [6] IMPLICIT SEQUENCE {
        rsp-serv                [0] IMPLICIT InitiateDownloadSequenceResponse,
        rsp-detail              [1] IMPLICIT CSInitiateDownloadSequenceResponse},
    download-segment          [7] IMPLICIT SEQUENCE {
        rsp-serv                [0] IMPLICIT DownloadSegmentResponse,
        rsp-detail              [1] IMPLICIT CSDownloadSegmentResponse},
    terminate-download-sequence [8] IMPLICIT SEQUENCE {
        rsp-serv                [0] IMPLICIT TerminateDownloadSequenceResponse,
        req-detail              [1] IMPLICIT CSTerminateDownloadSequenceResponse},
    initiate-upload-sequence   [9] IMPLICIT SEQUENCE {
        rsp-serv                [0] IMPLICIT InitiateUploadSequenceResponse,
        rsp-detail              [1] IMPLICIT CSInitiateUploadSequenceResponse},
    upload-segment            [10] IMPLICIT SEQUENCE {
        rsp-serv                [0] IMPLICIT UploadSegmentResponse,
        rsp-detail              [1] IMPLICIT CSUploadSegmentResponse},
    terminate-upload-sequence [11] IMPLICIT SEQUENCE {
        rsp-serv                [0] IMPLICIT TerminateUploadSequenceResponse,
        rsp-detail              [1] IMPLICIT CSTerminateUploadSequenceResponse},
    delete-domain             [12] IMPLICIT SEQUENCE {
        rsp-serv                [0] IMPLICIT DeleteDomainResponse,
        rsp-detail              [1] IMPLICIT CSDeleteDomainResponse},
--Services VMD
    status                    [15] IMPLICIT SEQUENCE {
        rsp-serv                [0] IMPLICIT StatusResponse,
        rsp-detail              [1] IMPLICIT CSStatusResponse},
    identify                  [16] IMPLICIT SEQUENCE {
        rsp-serv                [0] IMPLICIT IdentifyResponse,
        rsp-detail              [1] IMPLICIT CSIIdentifyResponse},
--Services PROGRAM INVOCATION--
    create-program-invocation [20] IMPLICIT SEQUENCE {
        rsp-serv                [0] IMPLICIT CreateProgramInvocationResponse,
        rsp-detail              [1] IMPLICIT CSCreateProgramInvocationResponse},
}

```

```

delete-program-invocation
  rsp-serv
  rsp-detail
start
  rsp-serv
  rsp-detail
stop
  rsp-serv
  rsp-detail
resume
  rsp-serv
  rsp-detail
reset
  rsp-serv
  rsp-detail
kill
  rsp-serv
  rsp-detail
--Services VARIABLE /VARIABLE-LIST--
read
  rsp-serv
  rsp-detail
write
  rsp-serv
  rsp-detail
define-variable-list
  rsp-serv
  rsp-detail
delete-variable-list
  rsp-serv
  rsp-detail
--Services EVENT--
get-alarm-summary
  rsp-serv
  rsp-detail
acknowledge-event-notification
  rsp-serv
  rsp-detail
alter-event-cond-monitor
  rsp-serv
  rsp-detail
--Services Get Attributes--
get-name-list
  req-serv
  req-detail
get-domain-attributes
  req-serv
  req-detail
get-program-invocation-attributes
  req-serv
  req-detail
get-variable-access-attributes
  req-serv
  req-detail
get-variable-list-attributes
  req-serv
  req-detail
get-event-condition-attributes
  req-serv
  req-detail
--Services OD--
generic-init-download
  req-serv
  req-detail
generic-download
  req-serv
  req-detail
generic-terminate-download
  req-serv
  req-detail
get-OD-header/datatype-attributes
  req-serv
  req-detail
}

[21] IMPLICITSEQUENCE {
[0] IMPLICIT DeleteProgramInvocationResponse,
[1] IMPLICIT CSDeleteProgramInvocationResponse},
[22] IMPLICITSEQUENCE {
[0] IMPLICIT StartResponse,
[1] IMPLICIT CSStartResponse},
[23] IMPLICITSEQUENCE {
[0] IMPLICIT StopResponse,
[1] IMPLICIT CSStopResponse},
[24] IMPLICITSEQUENCE {
[0] IMPLICIT ResumeResponse,
[1] IMPLICIT CSResumeResponse},
[25] IMPLICITSEQUENCE {
[0] IMPLICIT ResetResponse,
[1] IMPLICIT CSResetResponse},
[26] IMPLICITSEQUENCE {
[0] IMPLICIT KillResponse,
[1] IMPLICIT CSKillResponse},

[30] IMPLICITSEQUENCE {
[0] IMPLICIT ReadResponse,
[1] IMPLICIT CSReadResponse},
[31] IMPLICITSEQUENCE {
[0] IMPLICIT WriteResponse,
[1] IMPLICIT CSWriteResponse},
[35] IMPLICITSEQUENCE {
[0] IMPLICIT DefineVariableListResponse,
[1] IMPLICIT CSDefineVariableListResponse},
[36] IMPLICITSEQUENCE {
[0] IMPLICIT DeleteVariableListResponse,
[1] IMPLICIT CSDeleteVariableListResponse},

[39] IMPLICITSEQUENCE {
[0] IMPLICIT GetAlarmSummaryResponse,
[1] IMPLICIT CSGetAlarmSummaryResponse},
[40] IMPLICITSEQUENCE {
[0] IMPLICIT AcknowledgeEventNotificationResponse,
[1] IMPLICIT CSAcknowledgeEventNotificationResponse},
[41] IMPLICITSEQUENCE {
[0] IMPLICIT AlterEventConditionMonitoringResponse,
[1] IMPLICIT CSAAlterEventConditionMonitoringResponse},

[45] IMPLICITSEQUENCE {
[0] GetNameListResponse,
[1] IMPLICIT CSGetNameListResponse},
[46] IMPLICITSEQUENCE {
[0] IMPLICIT GetDomainAttributesResponse,
[1] IMPLICIT CSGetDomainAttributesResponse},
[47] IMPLICITSEQUENCE {
[0] IMPLICIT GetProgramInvocationAttributesResponse,
[1] IMPLICIT CSGetProgramInvocationAttributesResponse},
[48] IMPLICITSEQUENCE {
[0] IMPLICIT GetVariableAccessAttributesResponse,
[1] IMPLICIT CSGetVariablesAccessAttributesResponse},
[49] IMPLICITSEQUENCE {
[0] IMPLICIT GetVariableListAttributesResponse,
[1] IMPLICIT CSGetVariableListAttributesResponse},
[50] IMPLICITSEQUENCE {
[0] IMPLICIT GetEventConditionAttributesResponse,
[1] IMPLICIT CSGetEventConditionAttributesResponse},

[55] IMPLICITSEQUENCE {
[0] IMPLICIT GenericInitDownloadRequest,
[1] IMPLICIT CSGenericInitDownloadRequest},
[56] IMPLICITSEQUENCE {
[0] IMPLICIT GenericDownloadRequest,
[1] IMPLICIT CSGenericDownloadRequest},
[57] IMPLICITSEQUENCE {
[0] IMPLICIT GenericTerminateDownloadRequest,
[1] IMPLICIT CSGenericTerminateDownloadRequest},
[58] IMPLICITSEQUENCE {
[0] IMPLICIT GetODHeaderDataTypeRequest,
[1] IMPLICIT CSGetODHeaderDataTypeRequest},

```

ConfirmedServiceResp comprend trois paramètres:

- 1) [X] qui identifie le service,
- 2) rsq-serv qui fait référence à la production correspondant au service sélectionné,
- 3) rsp-detail qui fait référence à la production des CS correspondant au service sélectionné.

5.2.3.1.10 Définition d'error-type

```
ErrorType ::= CHOICE{
--Choix d'un sous-ensemble d'erreurs de la norme MMS. Les normes d'accompagnement peuvent allonger la liste
des erreurs de la norme MMS.--
--Erreurs de norme MMS:
    vmd                                [0] IMPLICIT UNSIGNED-16{
        other                          (0)},
    application-reference               [1] IMPLICIT UNSIGNED-16{
        other                          (0),
        application-unreachable        (1),
        -- valeur (2) réservée à la perte de connexion,
        application-reference-invalid   (3),
        context-unsupported             (4)},
    definition                         [2] IMPLICIT UNSIGNED-16 {
        other                          (0),
        object-undefined               (1),
        --valeur (2) réservée à l'adresse invalide--
        type-unsupported                (3),
        type-inconsistent              (4),
        object exist                    (5),
        object-attribute-inconsistent  (6)},
    resource                           [3] IMPLICIT UNSIGNED-16 {
        other                          (0),
        memory-unavailable              (1),
        --valeur (2) réservée à l'indisponibilité des ressources du processeur,--
        --valeur (3) réservée à l'indisponibilité de la mémoire de masse,--
        capability-unavailable          (4),
        capability-unknown              (5)},
    service                            [4] IMPLICIT UNSIGNED-16 {
        other                          (0),
        primitive-out-of-sequence       (1),
        object-state-conflict           (2)+256*"pi-state" value,
        --valeur (3) réservée à une autre définition,--
        --valeur (4) réservée à l'invalidité de la suite,--
        object-constraint-conflict      (5)+256*"pi-state" value},
    service-preempt                    [5] IMPLICIT UNSIGNED-16 {
        other                          (0),
        timeout                         (1),
        --valeur (2) réservée à l'interblocage,
        --valeur (3) réservée à l'annulation--},
    --valeur de type choice [6] réservée à la résolution temporelle,--
    access                             [7] IMPLICIT UNSIGNED-16 {
        other                          (0),
        object-access-unsupported       (1),
        object-non-existent            (2),
        object-access-denied           (3),
        object-invalidated             (4)},
    initiate                           [8] IMPLICIT UNSIGNED-16 {
        other                          (0),
        --valeurs (1) et (2) réservées à une autre définition--
        max serv outstanding calling insufficient (3),
        max serv outstanding called insufficient (4),
        service CBB insufficient        (5),
        parameter CBB insufficient      (6),
        nesting level insufficient      (7) },
    conclude                           [9] IMPLICIT UNSIGNED-16 {
        other                          (0),
        further-communication-required (1)},
    --choice value [10] reserved for cancel,--
    --valeur de type choice [11] réservée au fichier,--
    other                               [12] IMPLICIT UNSIGNED-16 {
        --valeurs réservées à l'initiative locale--
    }
}
```

En cas d'échec des services Start, Stop, Resume et Reset, Error-Type contient également des informations relatives à «class error» et à «code error» la valeur de «pi-state». En cas d'échec des autres services, il convient qu'Error-Type contienne des informations relatives à «class error» et «code error» et la valeur de «pi-state» est égale à (0).

NOTE 1 [x] représente la classe d'erreur; elle est identique aux MMS équivalents.

NOTE 2 (y) représente le code d'erreur, il est identique au MMS équivalent.

NOTE 3 Les valeurs autorisées de l'«état de pi» sont celles qui sont définies.

5.2.3.1.11 Définitions des types de données

Les spécifications des types de données échangées (données = contenu d'une variable ou «ErrorType») permettent de décrire la composition exacte des données, en indiquant le type et la longueur.

```
TypeDataSpecification ::= CHOICE{
--Les normes d'accompagnement et les utilisateurs Sub-MMS peuvent allonger cette liste.--
--Chaque extension de type doit être identifiée par [X] IMPLICIT «Y». Les valeurs de «X» variant de 15 à 25 sont
réservées pour les types définis par les Normes d'Accompagnement. Les valeurs de «X» variant de 26 à 127 sont
réservées pour les types définis par les utilisateurs. «Y» doit être défini en utilisant l'EN 50170 - Volume 3 de la
notation ASN.1 (voir annexe A)--
--TYPE DE DONNÉES--
--type de données construites --
    array-type                                [0] IMPLICIT SEQUENCE{
        number-of-elements                    UNSIGNED-8,
        element-type                          TypeDataSpecification
    }
--recommandé: nest=1--},
    structure-type                            [1] IMPLICIT SEQUENCE OF
        TypeDataSpecification
--recommandé: nest=1--},
--Type de données simples--
    Boolean-type                              [2] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 représente la longueur; la longueur est le nombre de bits utilisé pour coder la valeur de type
Boolean; les valeurs de longueur autorisées = 8.
    integer-type                             [3] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 représente la longueur; la longueur est le nombre de bits utilisé pour coder la valeur de type
integer; les valeurs de longueur autorisées = multiple de 8.--
    unsigned-type                            [4] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 représente la longueur; la longueur est le nombre de bits utilisé pour coder la valeur de type
unsigned; les valeurs de longueur autorisées = multiples de 8.--
    bitstring-type                           [5] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 représente la longueur, la longueur, est le nombre de bits utilisés pour coder la valeur de type
bitstring --
    visiblestring-type                       [6] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 représente la longueur; la longueur est le nombre d'octets utilisés pour coder la valeur de type
visiblestring.--
    octetstring-type                         [7] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 représente la longueur; la longueur est le nombre d'octets utilisés pour coder la valeur de type
octetstring.--
    time-difference                          [8] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 représente la longueur; la longueur est le nombre d'octets utilisé pour coder la valeur de type time-
difference; valeur de longueur autorisée = 4.--
    time-of-day-type                         [9] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 représente la longueur; la longueur est le nombre d'octets utilisé pour coder la valeur de type time-
of day; valeur de longueur autorisée = 6.--
    single-floating-type                     [10] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 représente la longueur; la longueur est le nombre d'octets utilisé pour coder la valeur de type
single floating; valeur de longueur autorisée = 4.--
    double-floating-type                     [11] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 représente la longueur; la longueur est le nombre d'octets utilisé pour coder la valeur de type
double floating; valeur de longueur autorisée = 8.--
    bcd-type                                [12] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 représente la longueur; la longueur est le nombre de chiffres--
    Boolean-array-type                       [13] IMPLICIT UNSIGNED-8,
--UNSIGNED-8 représente la longueur, la longueur est le nombre de bits utilisés pour coder la valeur de type
Boolean array.--
--ERROR-TYPE--
    error-type                               [14] IMPLICIT UNSIGNED-8
--UNSIGNED-8 représente la longueur; la longueur est le nombre d'octets utilisé pour coder la valeur de type error;
valeur de longueur autorisée = 3.--
}
```

5.2.3.1.12 Définitions des valeurs de données (sans longueur ou type)

- Les CS et les utilisateurs des services de Sub-MMS peuvent élargir cette liste de Valeurs de Données en utilisant la notation ASN.1.

- Les valeurs suivantes, résultant du codage des types de données ASN.1 et complétées par d'autres types de données jugées utiles, correspondent à des données dont le type et la longueur sont connus implicitement ou spécifiés par TypeDataSpecification ou explicitement spécifiés dans un service. Le type et la longueur ne figurent pas dans le codage, mais la valeur subsiste. Dans le cadre de la gestion des variables et des événements, le nombre d'octets utilisé pour coder la valeur doit correspondre à celle spécifiée par TypeDataSpecification. Ces données sont utilisées par exemple dans les services Read, Write, Information Report services, etc.--

DATA VALUE::= CHOICE{
 --Valeurs de données structurées--
 Array::= OCTETSTRING SIZE (n)
 --OCTETSTRING-SIZE (n) se compose de la juxtaposition des valeurs de données simples le constituant--
 Structure::= OCTETSTRING SIZE (n)
 --OCTETSTRING-SIZE (n) se compose de la juxtaposition des valeurs de données simples le constituant--
 --Valeurs de données simples--
 BOOLEAN::= BOOLEAN
 -- le codage est assuré sur 1 octet; la valeur binaire OOH signifie «false» et la valeur binaire ≠ 00H signifie «true»
 UNSIGNED-n::= Integer(0.. 2**n - 1)
 --«n» représente le nombre de bits utilisé; «n» est un multiple de 8--
 --le codage se fait sur une série de M octets avec M=n/8, le premier octet est appelé «octet LSB», le dernier octet est appelé «octet MSM». La valeur binaire est comprise entre: 0 and 2**n-1. Le MSB du nombre binaire est aligné avec le bit 8 du LSB de la chaîne d'octets et ainsi de suite--
 INTEGER-n::= Integer(-2**(n - 1)... +2**(n-1)-1)
 --«n» représente le nombre de bits utilisé; «n» est un multiple de 8--
 --la valeur de l'entier est codée sur une chaîne de M octets avec M=n/8, le premier octet est appelé « octet LSB », le dernier octet est appelé « octet MSM ». Le complément à deux de la valeur binaire est compris entre: 0: -2**(n-1) et + 2**(n-1)-1. Le signe de l'entier est aligné avec le bit 8 de l'octet LSB de la chaîne d'octets. Le MSB est aligné avec le bit 7 du premier octet et ainsi de suite--.
 BITSTRING-n::= BITSTRING SIZE(n)
 --«n» représente le nombre de bits utilisé; «n» est un nombre entier égal à 0, 1,--
 --le codage se fait sur une série de M octets avec M=n/8, le premier octet est appelé «octet LSB», le dernier bit est appelé «octet MSM». Le bit n° 0 est aligné avec le bit 8 de l'octet LSB de la chaîne d'octets et ainsi de suite--.
 TimeOfDay::= OCTET STRING SIZE(6)
 --le codage est réalisé sur une série de 6 octets. La valeur contient deux éléments d'informations juxtaposées, le premier indiquant le nombre de jours depuis 1984 codé en UNSIGNED-16, le deuxième indiquant le nombre de millisecondes depuis minuit, codé en UNSIGNED-32; l'octet de poids le plus fort d'UNSIGNED-16 est aligné avec l'octet de poids le plus faible de la chaîne d'octets. L'octet de poids le plus fort de UNSIGNED-32 est aligné avec l'octet LSB+2 de la chaîne d'octets--
 TimeDifference::= OCTET STRING SIZE(4)
 --le codage est réalisé sur une chaîne de 4 octets. La valeur indique le nombre de millisecondes depuis minuit codé en UNSIGNED-32; l'octet MSB d'UNSIGNED-32 est aligné avec le LSB de la chaîne d'octets--
 BCD-n::= OCTET STRING SIZE(M)
 --«n» représente le nombre de bits utilisé; «n» est un nombre entier égal à 0, 1, 2--
 --le codage est réalisé sur une série de M octets avec M=n/4. La valeur représente «n» chiffres. Chaque chiffre est codé en nombre binaire sur un quartet; le premier chiffre (de poids fort) est aligné avec l'octet de poids le plus fort de l'octet de poids le plus fort. Le quartet inutilisé, le cas échéant, doit être le quartet de poids le plus faible de l'octet de poids le plus fort. Les valeurs autorisées pour un quartet sont 0 à 9. Le MSB du quartet est aligné comme requis avec le bit 8 ou le bit 4.
 BooleanArray-n::= BITSTRING SIZE(n)
 --«n» représente le nombre de bits utilisé; «n» est un nombre entier égal à 0, 1, 2--
 --Le codage de Boolean Array est identique à celui de BITSTRING; il diffère seulement de par la dénomination des bits. Les bits sont désignés conformément aux règles suivantes : le bit n° 0 est aligné avec le bit 2**0 du premier octet, le bit n° 7 est aligné avec le bit 2**7 du premier octet, le bit n° 8 est aligné avec le bit 2**0 du deuxième octet, le bit n° 15 est aligné avec le bit 2**7 du deuxième octet, etc.--
 SingleFloating::= OCTET STRING SIZE(4)
 --le codage est réalisé sur une série de 4 octets conformément au format simple précision de la CEI 60559 -
 --
 DoubleFloating::= OCTET STRING SIZE(8)
 --«n» représente le nombre de bits utilisé; «n» est un nombre entier égal à 0, 1, 2, 3--
 --le codage est réalisé sur une série de 8 octets conformément au format double précision de la CEI 60559
 --
 OCTETSTRING-n::= OCTET STRING SIZE(n)
 --«n» représente le nombre de bits utilisé; «n» est un nombre entier égal à 0, 1, 2--
 --le codage est réalisé sur une chaîne de "n" octets.--
 VisibleString-n::= OCTET STRING SIZE(n)
 --le codage est identique à celui de la chaîne de "n" octets d'octets (OCTETSTRING-n) avec cependant les restrictions suivantes : chaque octet représente un caractère visible qui doit être choisi parmi les caractères de la norme MMS : "A" à "Z", "a" à "z", "\$", "_", "0" à "9"--

5.2.3.1.13 Définitions complémentaires

```

Identification ::= CHOICE {
    index                UNSIGNED-16 (0 .. 32K-1).
    name                 CHOICE {
        undefined-length-name [0] IMPLICIT OCTETSTRING SIZE(0 .. 32)
        cs-defined-length-name [5] IMPLICIT OCTETSTRING SIZE(X)—voir note
    }
}

```

NOTE CS-defined-length-name ne peut être utilisé que si les Normes d'Accompagnement ont défini la valeur $X \leq 32$. Dans ce cas, la longueur d'OCTETSTRING SIZE (X), n'est plus codée.

```

Discard ::= OCTET STRING SIZE(0 .. 3)
-- en cas de chargement d'un objet domain avec succès, son contenu est vide, dans le cas contraire, son
contenu est ErrorType. Discard est utilisé seulement dans «Terminate Download Sequence»--

```

```

ListOfAccessResult ::= BIT STRING
--chaque bit donne le résultat de l'accès à une variable. Un bit positionné à 1 signifie que l'accès est un
succès.--

```

```
ListOfObjectId ::= SEQUENCE OF Identification
```

```
ListOfObjectType ::= SEQUENCE OF TypeDataSpecification
```

```

ListOfObjectValue ::= OCTET STRING
--elle est la juxtaposition des valeurs des variables et/ou des types d'erreurs codées conformément à la «Data
Value» (Valeur de Données) ou à Error type

```

```

ListOfErrorType ::= OCTET STRING
--contient la juxtaposition de plusieurs valeurs ErrorType codées conformément à Error Type (Voir 5.2.3.1.10).

```

```

{
ObjectClass ::= UNSIGNED-8
    Null          (0),
    Domain        (1),
    PI            (2),
    variable      (3),
    Variable-List (4),
    Event         (5),
    OD Header     (6),
    Data Type     (7).
}

```

5.2.4 Protocole ASE VMD

5.2.4.1 Protocole de Gestion d'Environnement

5.2.4.1.1 Généralités

Le protocole de gestion d'association définit les PDU nécessaires à l'exécution des services suivants:

- 1) Initiate,
- 2) Conclude,
- 3) Abort,
- 4) Reject.

5.2.4.1.2 Définition du protocole du service confirmé initiate

NOTE Init Detail Request et Init Detail Response sont définis dans le module SUB-MMS-CORE,

```

InitiateReqPDU ::= SEQUENCE {
    quality-of-service-calling      QualityOfServiceCalling
    extension-calling              ExtensionCalling,
    proposed-max-serv-outstanding-calling INTEGER-16,
    proposed-max-serv-outstanding-called INTEGER-16,
    proposed-data-structure-nesting INTEGER-8,
    --nest=1 recommandé; doit être ≥ 1--
    init-detail-request            [0] IMPLICIT InitDetailRequest
}

```

```

InitiateRespPDU ::= SEQUENCE {
    quality-of-service-called          QualityOfServiceCalled,
    extension-called                  ExtensionCalled,
    negotiated-max-serv-outstanding-calling    INTEGER-16,
    negotiated-max-serv-outstanding-called    INTEGER-16,
    negotiated-data-structure-nesting    INTEGER-8,
    --nest=1 recommandé; doit être ≥ 1--
    init-detail-response                [0] IMPLICIT InitDetailResponse
    --voir 10.2.8--
}

QualityOfServiceCalling ::= BITSTRING-8 {
    quality-of-service                (0),
    --qualité de service 1 (valeur du bit = 1)--
    --quality de service 2 (valeur du bit = 0)--
}

ExtensionCalling ::= OCTETSTRING
    -- Pour la qualité 1 le contenu doit être vide--
    -- Pour la qualité 2 le contenu doit être AccessControl--
    -- Pour les extensions de qualité de service, le format de ce paramètre doit être spécifié.

QualityOfServiceCalled ::= BITSTRING-8 {
    quality-of-service                (0)
    -- qualité de service 1 (valeur du bit = 1): aucune information concernant cette qualité ne doit être incluse dans
    ExtensionCalling.--
    -- qualité de service 2 (valeur du bit = 0): les informations concernant cette qualité doivent être du type
    AccessControl; ces informations doivent être présentes dans ExtensionCalling et être incluses dans la
    première ligne (deux premiers octets du contenu OCTETSTRING)--
}

ExtensionCalled: : = OCTETSTRING
    }

AccessControl : : = BITSTRING-16 {
    password bit 8                    (0),
    password bit 7                    (1),
    password bit 6                    (2),
    password bit 5                    (3),
    password bit 4                    (4),
    password bit 3                    (5),
    password bit 2                    (6),
    password bit 1                    (7),

    access group 8                    (8),
    access group 7                    (9),
    access group 6                    (10),
    access group 5                    (11),
    access group 4                    (12),
    access group 3                    (13),
    access group 2                    (14),
    access group 1                    (15) }

InitiateErrorPDU ::= ErrorType

ParameterCBBSupportOption ::= BITSTRING{
    --valeur true=paramètre pris en charge. Les normes d'accompagnement peuvent allonger cette liste.--
    --types de données
    arr (array)                        (0),--bit N° 0
    str (structure)                    (1),--bit N° 1
    vlis (variable-list)                (2),--bit N° 2
    valt (partial access)                (3),--bit N° 3
}

ServiceSupportOption ::= BITSTRING {
    --Un service est pris en charge lorsque son bit prend la valeur true (1). Les normes d'accompagnement
    peuvent allonger cette liste en ajoutant de nouveaux services.
    a) le sous-ensemble de bits n° 0 à n° 39 indique la prise en charge des indications des services; ce sous-
    ensemble doit être présente dans BITSTRING,
    b) le sous-ensemble de bits de n° 40 à n° 79 indique la prise en charge des demandes de services et les bits
    s'étendant du n° 40 au n° 81 indiquent respectivement la prise en charge du nom pour les indications de
    services et les demandes de services; ce sous-ensemble est facultatif pour la demande et conditionnel pour la
    réponse.
    La longueur de la chaîne binaire peut donc prendre la valeur 39 ou la valeur 81--

```

--Les Bits N° 0 à 32 sont obligatoires--services de gestion de vmd pris en charge

status.ind (0), --bit N° 0
 unsolicited status.ind (non confirmé) (1), --bit N° 1
 identify.ind (2), --bit N° 2

-- services de gestion de domaine pris en charge

delete domain.ind (3), --bit N° 3
 initiate download sequence.ind (4), --bit N° 4
 download segment.ind (5), --bit N° 5
 terminate download sequence.ind (6), --bit N° 6
 initiate upload sequence.ind (7), --bit N° 7
 upload segment.ind (8), --bit N° 8
 terminate upload sequence.ind (9), --bit N° 9

-- Services de gestion de program invocations pris en charge

create program invocation.ind (10), --bit N° 10
 delete program invocation.ind (11), --bit N° 11
 start.ind (12), --bit N° 12
 stop.ind (13), --bit N° 13
 resume.ind (14), --bit N° 14
 reset.ind (15), --bit N° 15
 kill.ind (16), --bit N° 16

-- services de gestion de variable pris en charge

read.ind (17), --bit N° 17
 write.ind (18), --bit N° 18
 information report.ind (non confirmé) (19), --bit N° 19
 define variable-list.ind (20), --bit N° 20
 delete variable-list.ind (21), --bit N° 21

-- services de gestion d'event pris en charge

event notification.ind (non confirmé) (22), --bit N° 22
 alter event condition monitoring.ind (23), --bit N° 23
 acknowledge event notification.ind (24), --bit N° 24
 get alarm summary.ind (25), --bit N° 25

-- services de gestion de context pris en charge

conclude.ind (26), --bit N° 26

-- services de gestion d'attributs pris en charge

get name list.ind (27), --bit N° 27
 get domain attributes.ind (28), --bit N° 28
 get program invocation attributes.ind (29), --bit N° 29
 get variable access attributes.ind (30), --bit N° 30
 get variable-list attributes.ind (31), --bit N° 31
 get event condition attributes.ind (32), --bit N° 32

-- services OD

generic-init-download.ind (33), --bit N° 33
 generic-download.ind (34), --bit N° 34
 generic-terminate-download.ind (35), --bit N° 35
 get OD Header/Data Type.ind (36), --bit N° 36

--Les bits N°37 à 39 sont réservés aux extensions de services

--L'ensemble des bits 40 à 80 est facultatif pour les demandes et conditionnel pour les réponses

--services de gestion de vmd pris en charge

status.req (40), --bit N° 40
 unsolicited status.req (non confirmé) (41), --bit N° 41
 identify.req (42), --bit N° 42

-- services de gestion de domaine pris en charge

delete domain.req (43), --bit N° 43
 initiate download sequence.req (44), --bit N° 44
 download segment.req (45), --bit N° 45
 terminate download sequence.req (46), --bit N° 46
 initiate upload sequence.req (47), --bit N° 47
 upload segment.req (48), --bit N° 48
 terminate upload sequence.req (49), --bit N° 49

-- services de gestion de program invocation pris en charge

create program invocation.req (50), --bit N° 50
 delete program invocation.req (51), --bit N° 51
 start.req (52), --bit N° 52
 stop.req (53), --bit N° 53
 resume.req (54), --bit N° 54
 reset.req (55), --bit N° 55
 kill.req (56), --bit N° 56

-- services de gestion de variable pris en charge

read.req (57), --bit N° 57
 write.req (58), --bit N° 58
 information report.req (non confirmé) (59), --bit N° 59
 define variable-list.req (60), --bit N° 60
 delete variable-list.req (61), --bit N° 61

-- services de gestion d'event pris en charge

```

event notification.req non confirmé) (62), --bit N° 62
alter event condition monitoring.req (63), --bit N° 63
acknowledge event notification.req (64), --bit N° 64
get alarm summary.req (65), --bit N° 65
-- service de gestion de context pris en charge
conclude.req (66), --bit N° 66
-- services de gestion d'attributs pris en charge
get name list.req (67), --bit N° 67
get domain attributes.req (68), --bit N° 68
get program invocation attributes.req (69), --bit N° 69
get variable access attributes.req (70), --bit N° 70
get variable-list attributes.req (71), --bit N° 71
get event condition attributes.req (72), --bit N° 72
-- Services OD
generic-init-download.ind (73), --bit N° 73
generic-download.ind (74), --bit N° 74
generic-terminate-download.ind (75), --bit N° 75
get OD Header/Data Type.ind (76), --bit N° 76
--Les bits N°77 à 79 sont réservés aux extensions de services
--Type d'identification pris en charge
name support for service.ind (80), --bit N° 80
name support for service.req (81), --bit N° 81
}

```

5.2.4.1.3 Définition du protocole du service confirmé conclude

ConcludeReqPDU ::= NULL.

ConcludeRespPDU ::= NULL

ConcludeErrorPDU ::= ErrorType

5.2.4.1.4 Définition du protocole du service non confirmé abort

Le service Abort est mappé directement sur le service A-Abort de la couche inférieure pour assurer les connexions.

5.2.4.1.5 Définition du protocole du service non confirmé reject

Le service reject est décrit par la PDU Reject.

```

RejectPDU ::= SEQUENCE{
    original-invocation-Id          UNSIGNED-32,
    --non significatif si le rejet concerne Conclude--
    reject-reason                   RejectReason}
RejectReason ::= CHOICE {
    --Les normes d'accompagnement peuvent allonger cette liste --
    confirmed-request-PDU          [0] IMPLICIT UNSIGNED-8{
        other                      (0),
        unrecognized-service       (1),
        invalid-invocation-Id      (3),
        invalid-argument           (4),
        max-serv-outstanding-exceeded (6),
        max-recursion-exceeded     (8),
        value-out-of-range         (9)},
    pdu-error                      [4] IMPLICIT UNSIGNED-8{
        unknown-type-pdu          (0),
        invalid-pdu                (1),
        illegal-mapping            (2)},
    conclude-request-PDU          [8] IMPLICIT UNSIGNED-8{
        other                      (0),
        invalid-argument           (1)},
}

```

5.2.4.2 Protocole concernant la gestion de vmd

5.2.4.2.1 Généralités

Cette section définit le protocole des services suivants:

- 1) Status,

- 2) UnsolicitedStatus,
- 3) Identify,
- 4) Get Name List.

5.2.4.2.2 Définition du protocole du service confirmé status

- paramètres de demande.

StatusRequest ::= BOOLEAN --true pour la dérivation étendue

- paramètres de réponse.

```
StatusResponse ::= SEQUENCE{
  vmd-logical-status          UNSIGNED-8{
    state-change-allowed      (0),
    no-state-change-allowed    (1),
    others                      (2 to 5),
    --réservé aux états complémentaires. Ces états doivent être expliqués dans la déclaration PICS.
    Ces états complémentaires permettent d'assurer la compatibilité avec le MMS, FMS. Les valeurs 2 à 3 sont
    réservées pour assurer la compatibilité de Sub-MMS avec MMS. La valeur 2 est réservée pour l'état «limited
    service» autorisé. La valeur 3 est réservée à l'état «support service» autorisé. Les valeurs 4 à 5 sont réservées
    pour assurer la compatibilité de Sub-MMS avec FMS. La valeur 4 est réservée à l'état «loading of objects
    without consequences for the users». La valeur 5 est réservée à l'état «loading of objects with consequences
    for the users».
    (6 à n)
    --réservé aux C.S
  },
  vmd-physical-status         UNSIGNED-8{
    operational                (0),
    partially-operational       (1),
    inoperable                  (2),
    needs-commissioning         (3) },
  local-detail                 BITSTRING SIZE (0..128)
  --le contenu de type BIT STRING est vide si le détail local n'est pas utilisé --
}
```

- 1) «State change allowed» (valeur 0 pour UNSIGNED-8):
tous les services de Sub-MMS sont autorisés.
- 2) «No state change allowed» (valeur 1 pour UNSIGNED-8):
les services suivants sont autorisés.

Initiate,
Conclude,
Abort,
Status,
Identify.
Read
Get Alarm Summary
Get Name List
Get Domain Attributes
Get Program Invocation Attributes
Get Variable Access Attributes
Get Variable List Attributes
Get Event Condition Attributes

5.2.4.2.3 Définition du protocole du service non confirmé unsolicited status

- paramètres de service.

UnsolicitedStatus ::= StatusResponse

5.2.4.2.4 Définition du protocole du service confirmé identify

- paramètres de demande.

IdentifyRequest ::= NULL

– paramètres de réponse.

```
IdentifyResponse ::= SEQUENCE {
    vendor's name    VISIBLESTRING,
    model name       VISIBLESTRING,
    revision         VISIBLESTRING
}
```

5.2.4.2.5 Définition du protocole du service confirmé get name list

– paramètres de demande.

```
GetNameListRequest ::= SEQUENCE {
    object class      OBJECTCLASS {
        --values authorized
        null           (0),
        domain         (1),
        pi              (2),
        variable        (3),
        variable-list   (4),
        event           (5),
        ODHeader        (6),
        Datatype        (7).
    }
    continue after    identification
}
```

– paramètres de réponse.

```
GetNameListResponse ::= CHOICE {
    list-of-ID        SEQUENCE {
        --sélectionné si la valeur de la classe d'objets est différente de 0
        list-of-identification ListOfObjectId,
        -- si un objet prend en charge la double identification (nom et indice) seul l'indice sera transmis--
        more follows          BOOLEAN },
    list-of-description SEQUENCE {
        --sélectionné si la valeur de la classe d'objets est égale à 0
        object description     CHOICE {
            gives the description of the object
            domain-description  GetDomainAttributesResponse
            pi-description       GetProgramInvocationAttributesResponse
            variable-description GetVariableAccessAttributesResponse
            variable-list-description GetVariableListAttributesResponse
            event-description    GetEventConditionAttributesResponse
            od-header-description GetODHeader/DataTypeAttributesResponse
            data-type-description GetODHeader/DataTypeAttributesResponse
        }
        more follows          BOOLEAN
    }
}
```

5.2.5 Protocole d'ASE Domain

5.2.5.1 Généralités

Cette section définit les protocoles des services suivants:

- 1) DeleteDomain,
- 2) InitiateDownloadSequence,
- 3) DownloadSegment,
- 4) TerminateDownloadSequence,
- 5) InitiateUploadSequence,

- 6) UploadSegment,
- 7) TerminateUploadSequence,
- 8) Get Domain Attributes.

NOTE Le statut d'un «Domain» peut implicitement prendre les valeurs: «predefined/non-existent», «ready» et «in-use».

– un «domain» peut préexister ou peut être créé par un service de Sub-MMS.

– un «domain» préexistant peut être présenté avec ses données totales ou partielles, le reste de ses données peut être chargé par les services Sub-MMS.

– le service Delete Domain peut être exécuté localement.

5.2.5.2 Définition du protocole du service delete domain

- Paramètres de demande.

DeleteDomainRequest: = Identification.

- Paramètres de réponse.

DeleteDomainResponse::= NULL

5.2.5.3 Définition du protocole du service confirmé initiate download sequence

- Paramètres de demande.

```
InitDownloadSequenceRequest::= SEQUENCE{
    domain-identification      Identification,
    list-of-capabilities        SEQUENCE OF OCTETSTRING
-- le contenu peut être vide (attention: le codage de type SEQUENCE OF continue)
    sharable                   BOOLEAN--true for sharable
    domain-access-protection    DomainAccessProtection
}
```

NOTE Le paramètre «domain-access-protection» est affecté à l'objet Domain créé par le service. Il caractérise le niveau de protection de l'objet à l'égard des CLIENTS.

DomainAccessProtection::= BITSTRING{

--Pour un service de qualité 1, le contenu de BITSTRING doit être vide.

Pour un service de qualité 2, le contenu de BITSTRING doit correspondre à la définition suivante--

--mot de passe--

password-bit 8	(0),
password-bit 7	(1),
password-bit 6	(2),
password-bit 5	(3),
password-bit 4	(4),
password-bit 3	(5),
password-bit 2	(6),
password-bit 1	(7),

--groupe d'accès--

access-group 8	(8),
access-group 7	(9),
access-group 6	(10),
access-group 5	(11),
access-group 4	(12),
access-group 3	(13),
access-group 2	(14),
access-group 1	(15),

```
--protection d'accès à l'objet--
-- la valeur (16) est réservée--
    Ug                                (17),
--Si la valeur du bit = 1 et le client appartient à l'un des groupes spécifiés dans l'attribut «access group», alors
l'objet domain peut être attaché à une PI par le service Create PI--
    Wg                                (18),
--Si la valeur du bit = 1 et le client appartient à l'un des groupes spécifiés dans l'attribut «access group», l'objet
Domain peut être téléchargé/supprimé par les services Initiate Download Sequence, Download Segment,
Terminate Download Sequence et Delete Domain--
    Rg                                (19),
--Si la valeur du bit = 1 et le client appartient à l'un des groupes spécifiés dans l'attribut «groupe d'accès»,
alors l'objet Domain peut être chargé par les services Sequence, Upload Segment et Terminate Upload
Sequence--
-- la valeur (20) est réservée--
    U                                (21),
--Si la valeur du bit = 1 et le client a le mot de passe requis identique à l'attribut du «password», alors l'objet
Domain peut être attaché à une PI par le service Create PI--
    W                                (22),
--Si la valeur du bit = 1 et le client a donné le mot de passe requis identique à l'attribut «password», l'objet
Domain peut être téléchargé par les services Initiate Download Sequence, Download Segment, Terminate
Download Sequence et Delete Domain--
    R                                (23),
--Si la valeur du bit = 1 et le client a donné le mot de passe requis identique à l'attribut «password», l'objet
Domain peut être chargé par les services Initiate Upload Sequence, Upload Segment et Terminate Upload
Sequence--
--Les valeurs (24), (25), (26), (27), (28) sont réservées --
    Ua                                (29),
--Si la valeur du bit = 1, tous les clients peuvent attacher l'objet Domain à une PI par l'intermédiaire du service
Create PI--
    Wa                                (30),
--Si la valeur du bit = 1, tous les clients peuvent télécharger/supprimer l'objet Domain par les services Initiate
Download Sequence, Download Segment, Terminate Download Sequence et Delete Domain--
    Ra                                (31)
--Si la valeur du bit = 1, tous les clients peuvent alors charger l'objet Domain par les services Initiate Upload
Sequence, Upload Segment et Terminate Upload Sequence
}

```

– Paramètres de réponse.

InitDownloadSequenceResponse ::= NULL
Définition du protocole du service confirmé download segment

– Paramètres de demande

DownloadSegmentRequest ::= Identification.

– Paramètres de réponse

```
DownloadSegmentResponse ::= SEQUENCE{
    load-data                OCTETSTRING,
    more-follow              BOOLEAN
}
--Si la valeur est «true», cela signifie qu'il y a d'autres segments à télécharger--

```

Définition du protocole du service confirmé terminate download sequence

– Paramètres de demande.

```
TermDownloadSequenceRequest ::= SEQUENCE{
    domain-id                Identification,
    discard                  Discard
}
--le contenu de Discard est vide si le chargement du domaine est effectif, sinon il contient ErrorType
(attention: le codage de la longueur continue tant que le contenu de Discard est vide) voir 5.2.3.1.11.

```

– Paramètres de réponse.

TermDownloadSequenceResponse ::= NULL

Définition du protocole du service confirmé initiate upload sequence

- Paramètres de demande

InitUploadSequenceRequest ::= Identification.

- Paramètres de réponse

```

InitUploadSequenceResponse ::= SEQUENCE {
    ulsm Index                               Index,

    --Index relève du type UNSIGNED-16. Si l'index est significatif (la valeur de UNSIGNED-16 est ≤32K-1) il
    convient que la conservation soit poursuivie avec l'ulsm index; si, dans le cas contraire, l'index n'est pas significatif
    (la valeur de UNSIGNED-16 est 32K) la conservation doit continuer avec l'identifiant de «domain» (le Nom ou
    l'Index de l'objet Domain). La résolution de la non-ambiguïté entre Index de l'objet Domain et ULSM Index est à la
    charge du serveur.—

    list-of-capabilities                     SEQUENCE OF OCTETSTRING }

    --le contenu de la liste des capacités peut être vide (attention: le codage de SEQUENCE OF continue)—

```

5.2.5.4 Définition du protocole du service confirmé upload segment

- Paramètres de demande

UploadSegmentRequest ::= Identification

--Voir les contraintes ci-dessus du service Initiate Upload Sequence sur l'importance de l'«Identification».

«Identification» correspond soit au nom ou à l'Index de l'objet Domain ou à l'ULSM Index selon le contexte.--

- Paramètres de réponse

```
UploadSegmentResponse ::= SEQUENCE {
    load-data          OctetString,
    more-follow        BOOLEAN
-- «true» s'il existe d'autres segments à sauvegarder
}
```

5.2.5.5 Définition du protocole du service confirmé terminate upload sequence

- Paramètres de demande

TerminateUploadSequenceRequest ::= Identification

--Voir les contraintes du service Initiate Upload Sequence sur l'importance de l'"Identification".

«Identification» correspond soit au nom ou à l'Index de l'objet Domain ou à l'ULSM Index selon le contexte.--

- Paramètres de réponse

```
TerminateUploadSequenceResponse ::= NULL
```

5.2.5.6 Définition du protocole du service confirmé get domain attributes

- Paramètres de demande

GetDomainAttributesRequest ::= Identification

- Paramètres de réponse

```

GetDomainAttributesResponse ::= SEQUENCE {
    list-of-identification      SEQUENCE OF IDENTIFICATION
    object-class                ObjectClass
    --Valeur autorisée «Domain»
    list-of-capabilities        SEQUENCE OF OCTETSTRING,
    ----le contenu peut être vide (attention: le codage de SEQUENCE OF continue)--
    state                       DomainState,
    predefined                   BOOLEAN, --true if predefined--
    sub-mms-sharable            BOOLEAN, --true if sharable--
    list-of-pi                  ListOfObjectID
    upload-in-progress          UNSIGNED-8,
    --nombre de chargements en cours--
    domain-access-protection    DomainAccessProtection
    --si l'objet n'est pas protégé, son contenu doit être vide--
    extension                   OCTETSTRING}

DomainState ::= UNSIGNED-8
{
    non-existent/predefined    (0),
    loading                    (1),
    ready                       (2),
    in-use                      (3),
    complete                   (4),
    incomplete                  (5)
}
    
```

5.2.6 Protocole d'ASE Program Invocation

5.2.6.1 Généralités

Cette section définit le protocole relatif aux services suivants:

- 1) CreateProgramInvocation,
- 2) DeleteProgramInvocation,
- 3) Start,
- 4) Stop,
- 5) Resume,
- 6) Reset,
- 7) Kill,
- 8) Get program invocation attributes.

NOTE Les 7 premiers services peuvent être exécutés localement.

5.2.6.2 Définition du protocole du service confirmé create program invocation

– Paramètres de demande

```

CreateProgramInvocationRequest ::= SEQUENCE{
    pi-identification            Identification,
    list-of-domains-identification ListOfObjectID,
    reusable                     BOOLEAN,
    --«true» pour réutilisable--
    pi-access-protection         PiAccessProtection
    extension                    OCTETSTRING}
    
```

NOTE Le paramètre «pi-access-protection» est affecté à l'objet PI créé par le service. Il caractérise le niveau de protection de l'objet à l'égard des CLIENTS.

```

PiAccessProtection ::= BITSTRING{
  --Pour une qualité du service 1, le contenu de BITSTRING doit être vide
  Pour une qualité du service 2, le contenu de BITSTRING doit correspondre à la définition suivante--
  --mot de passe--
  password-bit 8          (0),
  password-bit 7          (1),
  password-bit 6          (2),
  password-bit 5          (3),
  password-bit 4          (4),
  password-bit 3          (5),
  password-bit 2          (6),
  password-bit 1          (7),
  --groupe d'accès--
  access-group 8          (8),
  access-group 7          (9),
  access-group 6          (10),
  access-group 5          (11),
  access-group 4          (12),
  access-group 3          (13),
  access-group 2          (14),
  access-group 1          (15),
  --protection d'accès à l'objet--
  -- la valeur (16) est réservée--
  Dg                      (17),
  --Si la valeur du bit = 1 et le client appartient à l'un des groupes spécifiés dans l'attribut «access groups», alors
  il peut effacer la PI à l'aide des services Kill, Delete-PI--
  Hg                      (18),
  --Si la valeur du bit = 1 et le client appartient à l'un des groupes spécifiés dans l'attribut «access groups»,
  alors, il peut arrêter l'exécution de la PI par le service Stop --
  Sg                      (19),
  --Si la valeur du bit = 1 et le client appartient à l'un des groupes spécifiés dans l'attribut «access groups»,
  alors, il peut démarrer l'exécution de la PI par les services Start, Resume et Reset --
  -- la valeur (20) est réservée--
  D                      (21),
  --Si la valeur du bit = 1 et le client a fourni un mot de passe identique à celui de l'attribut «password» il peut
  effacer la PI par les services Kill, Delete-PI--
  H                      (22),
  --Si la valeur du bit = 1 et le client a fourni un mot de passe identique à celui de l'attribut «password», alors, il
  peut arrêter l'exécution de la PI par le service Stop--
  S                      (23),
  --Si la valeur du bit = 1 et le client a fourni un mot de passe identique à celui de l'attribut «access group», il
  peut démarrer l'exécution de la PI par les services Start, Resume et Reset--
  --Les valeurs (24), (25), (26), (27), (28) sont réservées--
  Da                      (29),
  --Si la valeur du bit = 1, tous les clients peuvent effacer la PI par les services Kill, Delete-PI--
  Ha                      (30),
  --Si la valeur du bit = 1, tous les clients peuvent arrêter l'exécution de la PI par le service Stop--
  Sa                      (31),
  --Si la valeur du bit = 1, tous les clients peuvent démarrer l'exécution de la PI par les services Start, Resume
  et Reset--
}
}

```

– Paramètres de réponse

CreateProgramInvocationResponse ::= Index

Le serveur peut informer le client qu'il accepte d'attribuer l'identifiant proposé à l'objet créé en renvoyant dans sa réponse un index de type UNSIGNED_16 avec une valeur non significative (Valeur = 32K).

Le serveur peut indiquer au client qu'il attribue à l'objet créé un identifiant du type d'index en renvoyant dans sa réponse un index avec une valeur significative (Valeur entre 0 et 32K-1); cet identifiant remplace l'identifiant proposé s'il est du type de l'index, sinon, l'objet créé prend en charge le nom et l'index comme identifiant.--

5.2.6.3 Définition du protocole du service confirmé delete program invocation

– Paramètres de demande.

DeleteProgInvocRequest ::= Identification.

– Paramètres de réponse.

DeleteProgInvocResponse ::= NULL

5.2.6.4 Définition du protocole du service confirmé start

- Paramètres de demande

StartRequest ::= SEQUENCE{
 pi-identification Identification,
 execution-argument OCTETSTRING }
 --le contenu d'execution argument peut être vide--

- Paramètres de réponse

StartResponse ::= NULL

5.2.6.5 Définition du protocole du service confirmé stop

- Paramètres de demande

StopRequest ::= Identification

- Paramètres de réponse

StopResponse ::= NULL

5.2.6.6 Définition du protocole du service confirmé resume

- Paramètres de demande

ResumeRequest ::= SEQUENCE{
 pi-identification Identification,
 execution-argument OCTETSTRING }
 --le contenu d'execution argument peut être vide--

- Paramètres de réponse

Resume Response ::= NULL

5.2.6.7 Définition du protocole du service confirmé reset

- Paramètres de demande

ResetRequest ::= Identification

- Paramètres de réponse

ResetResponse ::= NULL

5.2.6.8 Définition du protocole du service confirmé kill

- Paramètres de demande

KillRequest ::= Identification

- Paramètres de réponse

KillResponse ::= NULL

5.2.6.9 Définition du protocole du service confirmé get program invocation attributes

- Paramètres de demande

GetProgramInvocationAttributesRequest:: = Identification

– Paramètres de réponse

```

GetProgramInvocationAttributesResponse::= SEQUENCE {
    list-of-identification      SEQUENCE OF Identification
    object-class               ObjectClass --Valeur autorisée «PI»--
    pi-state                   PIState
    list-of-domain-id          ListOfObjetid
    sub-mms-deletable         BOOLEAN, --«true» si effaçable
    reusable                   BOOLEAN, --«true» si réutilisable
    execution-argument         OCTETSTRING,
    --le contenu d'OCTETSTRING peut être vide, sinon il convient qu'il contienne la dernière valeur fournie par le
    service Start ou Resume exécuté avec succès--
    pi-access-protection       PIAccessProtection
    --il convient que le contenu soit vide si l'objet n'est pas protégé
    extension                  OCTETSTRING }

PISTATE::= UNSIGNED-8 {
    non-existent               (0)
    unrunnable                 (1)
    idle                       (2)
    running                    (3)
    stopped                    (4)
    starting                   (5),
    stopping                   (6)
    resuming                   (7)
    resetting                  (8)
}

```

5.2.7 Protocole d'ASE Variable

5.2.7.1 Résumé

Cette section définit les paramètres de service suivants:

- 1) Read (une variable-list),
- 2) Write (une variable-list),
- 3) InformationReport (une variable-list),
- 4) DefineVariableList (une variable-list),
- 5) DeleteVariableList (une variable-list),
- 6) Get Variable Access Attributes (une variable),
- 7) Get Variable List Attributes (une variable-list).

5.2.7.2 Définitions communes

La production suivante est utilisée par les services Read (sans résultat), Write et Information Report--

```

VariableDescription::= CHOICE {
    variable_list [0]          IMPLICIT Identification,
    list of variables           [1] IMPLICIT SEQUENCE OF VariableAccessDescription
}

```

--La production suivante est uniquement utilisée par le service Read (avec résultat)--

```

VariableDescriptionWithType::= CHOICE {
    variable_list [2]          IMPLICIT Identification,
    list of variables           [3] IMPLICIT SEQUENCE OF VariableAccessDescription
}

```

--La production suivante est utilisée par les services Read, Write et Information Report. Elle permet un accès global ou partiel à des variables identifiés par leur index ou leur nom--

```

VariableAccessDescription ::= CHOICE {
--choix d'accès global
    object-index          UNSIGNED-16 (0..32K-1),
    object-name           CHOICE {
        undefined-length-name [0] IMPLICIT OCTETSTRING SIZE (0..32),
        CS-fixed-length-name  [5] IMPLICIT OCTETSTRING SIZE (X)-- voir note 1
    }
--choix d'accès partiel
--liste de sélection des comp/élém
    object-index/list-of-selection [(n≥1) th level nesting]
    object-index                  [1] IMPLICIT SEQUENCE {
        UNSIGNED-16 (0..32K-1),
        list-of-selection [0] IMPLICIT Selections },
    object-name/list-of-selection CHOICE {
        undefined-length-name/list-of-selection [2] IMPLICIT SEQUENCE {
            OCTETSTRING SIZE (0..32),
            list-of-selection [0] IMPLICIT Selections },
        cs-fixed-length-name/list-of-selection [6] IMPLICIT SEQUENCE {
            OCTETSTRING SIZE (X)—voir note 1
            list-of-selection [0] IMPLICIT Selections },
        },
    --one-comp/elem-selection
    object-index/one-selection [3] IMPLICIT SEQUENCE {
        object-index UNSIGNED-16 (0..32K-1),
        one-selection  UNSIGNED-16 },
    object-name-one-selection CHOICE {
        undefined-length-name/one-selection [4] IMPLICIT SEQUENCE {
            OCTETSTRING SIZE (0..32),
            one-selection  UNSIGNED-16 },
        cs-fixed-length-name/one-selection [7] IMPLICIT SEQUENCE {
            OCTETSTRING SIZE (X)—voir note 1
            one-selection  UNSIGNED-16 },
        },
    }
}

```

NOTE 1 CS-fixed-length-name ne peut être utilisé que si les Normes d'Accompagnements ont défini une valeur à X (X ≤ 32). Dans ce cas, la longueur d'OCTETSTRING SIZE (X) n'est plus codée.

```

Selections ::= SEQUENCE {
    alternate access          SEQUENCE OF {--le numéro de l'index doit être ≥ 0.--
        successive deeper comp/elem index UNSIGNED-16
    }
    access CHOICE {
        list of comp/elem index [0] IMPLICIT SEQUENCE OF UNSIGNED-16
        range of elements [1] IMPLICIT SEQUENCE {
            start element index UNSIGNED-16
            number of elements  UNSIGNED-16
        }
    }
}

```

NOTE 2 Un tableau est composé de plusieurs éléments. Chaque élément peut être de type simple, tableau ou structure. Chaque élément est identifié par un index de type UNSIGNED-16. Le premier élément est identifié par un index de valeur 0, le deuxième avec la valeur 1, etc. La composition d'un tableau est supposée être connue de tous les clients.

NOTE 3 Une structure est composée de plusieurs champs. Chaque champ peut être de type simple, tableau ou structure. Chaque champ est identifié par un index de type UNSIGNED-16. Aucune règle n'est imposée aux valeurs à attribuer aux identificateurs de champ. La composition d'une structure et les valeurs des identificateurs de champ sont supposées être connues de tous les clients.

5.2.7.3 Définition du protocole du service confirmé read

– Paramètres de demande

```

ReadRequest ::= CHOICE {
  without result variable access specification  VariableDescription ,
    --Ce choix indique une liste de variables à lire. Il indique également que la réponse ne doit pas contenir
    les descripteurs des types de valeurs lues--
  with result variable access specification  VariableDescriptionWithType
    --Ce choix indique une liste de variables à lire. Il indique également que la réponse doit contenir les
    descripteurs du type des variables--
}

```

– Paramètres de réponse.

-- La procédure de lecture pour une variable exécutée avec succès entraîne le retour de la valeur de la variable. Un échec de la procédure de lecture pour une variable entraîne le retour de la valeur ErrorType--

```

ReadResponse ::= SEQUENCE {
  list-of-access-result  ListOfAccessResult,
    --ListOfAccessResult est du type BITSTRING. Bit (0) est attribué à la première variable lue, bit (1) à
    la deuxième variable lue et ainsi de suite. Pour chaque variable lue, le bit qui lui est attribué indique
    si l'opération de lecture a été exécutée avec succès ou a échoué. Un bit mis à 0 indique que la
    tentative de lecture a échoué--
  list-of-Type-Data-Specif.  ListOfObjectType,
    --ListOfObjectType ne contient aucun descripteur si la demande ne nécessite pas leur restitution.
    Dans d'autres cas, il est composé de descripteurs des variables de lecture. Si la lecture d'une
    variable est exécutée avec succès, il convient que le descripteur indique le type de la valeur lue. Au
    contraire, si elle se termine par un échec, il convient que le descripteur indique le type de la valeur
    ErrorType.
    Si toutes les lectures des variables sont exécutées avec succès, et que les valeurs lues sont du
    même type, ListOfObjectType peut alors contenir un descripteur unique qui décrit le type en
    question. Les types autorisés sont ceux spécifiés par «TypeDataSpecification» (voir la section
    5.2.3.1.11)--
  list-of-Data-value  ListOfObjectValue
    --Le contenu de ListOfObjectValue est composé des données des variables lues. Ces données sont
    juxtaposées. Si la lecture d'une variable est exécutée avec succès, il convient que les données
    soient de la valeur lue. (voir 5.2.3.1.12 «Valeur de données» pour les différentes valeurs possibles).
    Si la lecture d'une variable se termine par un échec, les données doivent indiquer la valeur
    ErrorType. (voir 5.2.3.1.10)--
}

```

5.2.7.4 Définition du protocole du service confirmé write

– Paramètres de demande.

```

WriteRequest ::= SEQUENCE{
  variables access specification  VariableDescription,
    --VariableDescription indique une liste d'identifiants d'objets de classe variable ou un identifiant d'objets
    de classe variable-list (liste de variables)--
  list-of-Type-Data-Specif.  ListOfObjectType,
    --Le contenu de ListOfObjectType a pu être exempt de descripteur de type.
    Si non, il convient qu'il contienne le descripteur de chacune des valeurs des variables à écrire.
    S'il contient un seul descripteur, cela signifie que les valeurs des variables à écrire sont toutes du
    type indiqué par le descripteur. Les types autorisés sont ceux spécifiés par «TypeDataSpecification»
    (voir 5.2.3.1.11)--
  list-of-Data-Value  ListOfObjectValue
    --Le contenu de ListOfObjectValue doit correspondre aux valeurs des variables à écrire. Ces valeurs
    sont juxtaposées. Les valeurs autorisées sont celles spécifiées par «Data Value» (voir 5.2.3.1.2)--
}

```

– Paramètres de réponse.

```

WriteResponse ::= SEQUENCE {
  list-of-access-result  ListOfAccessResult,
    --ListOfAccessResult est du type BITSTRING. Le bit (0) est attribué à la première variable écrite, le bit
    (1) est attribué à la deuxième variable écrite et ainsi de suite. Pour chaque variable écrite, le bit qui lui
    est attribué indique si l'opération d'écriture a été exécutée avec succès ou a échoué. Un bit mis à 0
    indique que l'opération d'écriture a échoué--
  list-of-ErrorType  ListOfErrorType
    --ListOfErrorType ne contient aucune valeur si toutes les opérations d'écriture des variables ont été
    exécutées avec succès. Si non, il contient la juxtaposition des valeurs ErrorType de toutes les écritures
    de variables qui se sont terminées par un échec. Il convient que la première valeur corresponde au
    premier échec, la deuxième au deuxième échec et ainsi de suite (pour la valeur ErrorType Voir
    5.2.3.1.10)--
}

```

5.2.7.5 Définition du protocole du service non confirmé information report

– Paramètres de demande

```

InformationReportRequest ::= SEQUENCE{
    local detail OCTETSTRING,
    --le contenu de l'OCTETSTRING peut être vide--
    variable access specification VariablesDescription,
    --VariableDescription indique une liste des identifiants d'objets de classe variable ou un identifiant
    d'objet de classe variable-list (liste de variables) dont les valeurs sont fournies dans le paramètre
    ListOfObjectValue--
    list-of-Type-Data-Specif. ListOfObjectType,
    --le contenu de ListOfObjectType pouvait être vide.
    Autrement, il est composé de descripteurs des valeurs présentes dans le paramètre ListOfObjectValue.
    S'il contient un seul descripteur, cela signifie que toutes les valeurs du paramètre «ListOfObjectValue»
    sont du même type. Les types autorisés sont ceux spécifiés par «TypeDataSpecification» à l'exception
    de type d'erreur (voir 5.2.3.1.10)--
    list-of-Data-Value ListOfObjectValue }

    --Le contenu de ListOfObjectValue est composé des données des variables identifiées par le paramètre
    «VariablesDescription». Ces données sont juxtaposées. Les valeurs autorisées sont celles spécifiées
    par «Valeur de Données» (voir 5.2.3.1.11)--

```

5.2.7.6 Définition du protocole du service confirmé define variable list

– Paramètres de demande

```

DefineVariableListRequest ::= SEQUENCE{
    proposed-Var-list-Identification Identification,
    --l'identifiant proposé peut être un nom ou un Index.--
    list-of-var-Identification SEQUENCE OF Identification
    --Les identifiants des variables (pas de liste de variables) à attacher à la liste de variables créée--
    variable-list access protection VariableListAccessProtection,
    extension OCTETSTRING
    --permet de caractériser le type d'utilisateur, l'extension, etc.--
}

```

Révision:

- Le paramètre «variable-list access protection» est attribué à l'objet créé Variable-list. Il caractérise le niveau de protection de l'objet à l'égard des CLIENTS.

```

VarListAccessProtection ::= BITSTRING{
  --pour un service de qualité 1, il convient que le contenu de BITSTRING soit vide.
  Pour un service de qualité 2, il convient que le contenu de BITSTRING corresponde à la définition suivante--
  --mot de passe--
  password-bit 8,
  password-bit 7
  password-bit 6
  password-bit 5
  password-bit 4
  password-bit 3
  password-bit 2
  password-bit 1
  (1),
  (2),
  (3),
  (4),
  (5),
  (6),
  (7),

  --groupe d'accès--
  access-group 8
  access-group 7
  access-group 6
  access-group 5
  access-group 4
  access-group 3
  access-group 2
  access-group 1
  (8),
  (9),
  (10),
  (11),
  (12),
  (13),
  (14),
  (15),

  --protection de l'accès à l'objet--
  -- la valeur (16) est réservée--
  Dg
  --Si la valeur du bit = 1 et le client possède les droits requis «access group», il peut effacer l'objet Variable-list
  à l'aide du service Delete Variable-list--
  Wg
  --Si la valeur du bit = 1 et le client possède les droits requis «access group», il peut écrire dans les variables
  attachées à la variable -list à l'aide du service Write (variable-list) --
  Rg
  --Si la valeur du bit = 1 et le client possède les droits requis «access group», il peut lire les variables attachées
  à la variable-list à l'aide du service Read (Variable-list)--
  -- la valeur (20) est réservée--
  D
  --Si la valeur du bit = 1 et le client possède le droit requis «password», il peut effacer l'objet variable-list à
  l'aide du service Delete variable-list--
  W
  --Si la valeur du bit = 1 et le client possède les droits requis «password», il peut écrire dans les variables
  attachées à la variable -list à l'aide du service Write (Variable-list) --
  R
  --Si la valeur du bit = 1 et le client possède le droit requis «access group», il peut lire les variables attachées à
  la variable-list à l'aide du service Read (variable-list)--
  --Les valeurs (24), (25), (26), (27), (28) sont réservées--
  Da
  --Si la valeur du bit = 1, tous les clients peuvent effacer l'objet variable list à l'aide du service Delete variable-
  list--
  Wa
  --Si la valeur du bit = 1, tous les clients peuvent écrire dans les variables attachées à la variable-list à l'aide du
  service Write (Variable-list) --
  Ra
  --Si la valeur du bit = 1, tous les clients peuvent lire les variables attachées à la variable-list par l'intermédiaire
  du service Read(Variable-list)--
}

-- Paramètres de réponse

DefineVariableListResponse ::= Index

```

Le serveur peut indiquer au client qu'il accepte d'attribuer l'identifiant proposé à l'objet créé en renvoyant dans sa réponse un index ayant une valeur non significative (Valeur = 32K).

Le serveur peut indiquer au client qu'il attribue à l'objet créé un identifiant du type d'index à l'objet créé en renvoyant dans sa réponse un index avec une valeur significative (Valeur entre 0 et 32K-1); cet identifiant remplace l'identifiant proposé s'il est du type de l'index, sinon, l'objet créé prend en charge le nom et l'index comme identifiant.--

5.2.7.7 Définition du protocole du service confirmé delete variable list

– Paramètres de demande.

DeleteVariableListRequest ::= Identification.

Il convient qu'"Identification" soit l'identifiant de l'objet créé variable-list. Le type d'identification d'origine peut être un Nom, ou un Index ou les deux. Voir les explications données par le service DEFINE VARIABLE-LIST.--

- Paramètres de réponse.

DeleteVariableListResponse::= NULL

5.2.7.8 Définition du protocole du service confirmé get variable access attributes

- Paramètres de demande.

GetVariableAccessAttributesRequest::= Identification.

- Paramètres de réponse.

GetVariableAccessAttributesResponse::= SEQUENCE	{
list-of-identification	SEQUENCE OF IDENTIFICATION
object-class	Object-Class
	--Valeur autorisée «variable»--
type data description	TypeDataSpecification,
variableaccessprotection	VariableAccessProtection
extensionOCTETSTRING	}

VariableAccessProtection ::= BITSTRING{
 --le contenu doit être vide si l'objet n'est pas protégé, sinon, il convient qu'il contienne les informations
 définies ci-dessous--

--mot de passe--

password-bit 8	(0),
password-bit 7	(1),
password-bit 6	(2),
password-bit 5	(3),
password-bit 4	(4),
password-bit 3	(5),
password-bit 2	(6),
password-bit 1	(7),

--groupe d'accès--

access-group 8	(8),
access-group 7	(9),
access-group 6	(10),
access-group 5	(11),
access-group 4	(12),
access-group 3	(13),
access-group 2	(14),
access-group 1	(15),

--protection de l'accès à l'objet--

--valeurs (16), (17) réservées--

Wg	(18),
----	-------

--Si la valeur du bit = 1 et le client appartient à l'un des groupes spécifiés dans l'attribut «group access», il peut écrire dans la variable à l'aide du service Write (variable) --

Rg	(19),
----	-------

--Si la valeur du bit = 1 et le client appartient à l'un des groupes spécifiés dans l'attribut «access group», il peut lire la variable à l'aide du service Read (Variable)--

--valeurs (20), (21) sont réservées--

W	(22),
---	-------

--Si la valeur du bit = 1 et le client possède les droits requis «password», il peut écrire dans la variable à l'aide du service Write (variable) --

R	(23),
---	-------

--Si la valeur du bit = 1 et le client possède les droits requis «group access», il peut lire le contenu de la variable à l'aide du service Read (Variable) --

--Les valeurs (24), (25), (26), (27), (28), (29) sont réservées --

Wa	(30),
----	-------

--Si la valeur du bit = 1, tous les clients peuvent écrire dans la variable à l'aide du service Write (variable) -

Ra	(31)
----	------

--Si la valeur du bit = 1, tous les clients peuvent lire la variable à l'aide du service Read (Variable)--

extension OCTETSTRING}

5.2.7.9 Définition du protocole du service confirmé get variable list attributes

– Paramètres de demande.

GetVariableListAttributesRequest ::= Identification.

– Paramètres de réponse.

```
GetVariableListAttributesResponse ::= SEQUENCE {
    list-of-identification      SEQUENCE OF IDENTIFICATION
    object-class                ObjectClass
    sub-mms-deletable          BOOLEAN, --true if deletable
    list-of-variable            ListOfObjectID,
    variable-list-access-protection VariableListAccessProtection
    --il convient que le contenu soit vide si l'objet n'est pas protégé.--
    extension                  OCTETSTRING }
```

```
VariableListAccessProtection ::= BITSTRING{
    --le contenu doit être vide si l'objet n'est pas protégé, sinon, il convient qu'il contienne les informations
    définies ci-dessous--
    --mot de passe--
    password-bit 8           (0),
    password-bit 7           (1),
    password-bit 6           (2),
    password-bit 5           (3),
    password-bit 4           (4),
    password-bit 3           (5),
    password-bit 2           (6),
    password-bit 1           (7),
    --groupe d'accès--
    access-group 8           (8),
    access-group 7           (9),
    access-group 6           (10),
    access-group 5           (11),
    access-group 4           (12),
    access-group 3           (13),
    access-group 2           (14),
    access-group 1           (15),
    --protection de l'accès à l'objet--
    -- la valeur (16) est réservée--
    Dg                       (17),
    --Si la valeur du bit = 1 et le client appartient à l'un des groupes spécifiés dans l'attribut «Access Groups»,
    alors il peut effacer l'objet Variable-List à l'aide du service Delete Variable-List--
    Wg                       (18),
    --Si la valeur du bit = 1 et le client appartient à l'un des groupes spécifiés dans l'attribut «Access Groups»,
    alors il peut écrire dans la variable à l'aide du service Write (Variable-List) --
    Rg                       (19),
    --Si la valeur du bit = 1 et le client appartient à l'un des groupes spécifiés dans l'attribut «Access Groups»,
    alors il peut lire la variable à l'aide du service Read (Variable-List)--
    -- la valeur (20) est réservée--
    D                        (21),
    --Si la valeur du bit = 1 et le client a donné un mot de passe identique à celui spécifié dans l'attribut
    «Password», il peut effacer l'objet Variable-List à l'aide du service Delete Variable-List--
    W                        (22),
    --Si la valeur du bit = 1 et le client a donné un mot de passe identique à celui spécifié dans l'attribut
    «Password», il peut écrire dans la variable à l'aide du service Write (Variable-List).
    R                        (23),
    --Si la valeur du bit = 1 et le client a donné un mot de passe identique à celui spécifié dans l'attribut
    «Password», il peut lire la variable à l'aide du service Read (Variable-List)--
    --Les valeurs (24), (25), (26), (27), (28) sont réservées--
    Da                       (29),
    --Si la valeur du bit = 1, tous les clients peuvent effacer l'objet Variable-List à l'aide du service Delete Variable-
    List--
    Wa                       (30),
    --Si la valeur du bit = 1, tous les clients peuvent écrire dans la variable à l'aide du service Write (Variable list) -
    Ra                       (31),
    --Si la valeur du bit = 1, tous les clients peuvent lire la variable à l'aide du service Read (Variable list)--
    }
```

5.2.8 Protocole d'ASE Event

5.2.8.1 Généralités

Cette section définit le protocole pour les services suivants:

- 1) Event Notification,
- 2) Acknowledge Event Notification,
- 3) Alter Event Condition Monitoring,
- 4) Get Alarm Summary,
- 5) Get Event Condition Attributes

5.2.8.2 Définition du protocole du service non confirmé event notification

– Paramètres de demande.

EventNotification ::= SEQUENCE {
common-information CommonInformation
list-of-event-identification ListOfObjectId,
list-of-event-message-description SEQUENCE OF TypeDataSpecification
--NOTE
--Le paramètre "list-of-event-message-description" donne la liste des descripteurs des attributs
«EventMessage» des objets énumérés dans le paramètre ListOfObjectId. Cette liste pourrait être réduite à
ZÉRO descripteur. Chaque TypeDataSpecification décrit le type d'EventMessage.
Chaque «Event Message» est une structure ayant un ou plusieurs champs en fonction du type de l'objet
event.
Selon le type de l'objet, la description d'un «EventMessage» doit être celui d'une structure ayant:
a) soit uniquement un seul champ:
- ce champ décrit Event Data. Les types autorisés sont ceux définis par TypeDataSpecification à l'exception
d'error type.
b) soit deux champs:
- le premier champ décrit Event Occurrence ID. Le type autorisé est OCTETSTRING SIZE(2).
- le second champ décrit Event Data. Les types autorisés sont ceux définis par TypeDataSpecification à
l'exception d'error type.
b) soit trois champs:
- le premier champ décrit Event State. Le type autorisé est BITSTRING-16.
- le deuxième champ décrit Event Occurrence ID. Le type autorisé est OCTETSTRING SIZE(2).
- le troisième champ décrit Event Data. Les types autorisés sont ceux définis par TypeDataSpecification à
l'exception d'error type.
Si tous les attributs «EventMessage» sont du même type, la liste pourrait être réduite à un seul descripteur
«Event Message»--
list-of-event-message-value OCTETSTRING
--Ce paramètre contient la juxtaposition des valeurs d'EventMessage.
NOTE
En fonction du type d'objet, la valeur d'un «EventMessage» comprend:
a) un seul champ:
ce champ donne la valeur Event Data. Les valeurs autorisées sont celles définies par Valeur de Données
(voir 5.2.3.1.12).
b) ou deux champs:
- le premier donne la valeur d'Event Occurrence ID. La valeur autorisée est OCTETSTRING-2,
- Le second donne la valeur d'Event Data. Les valeurs autorisées sont celles définies par Valeur de
Données (voir 5.2.3.1.2),
b) ou trois champs:
- le premier donne la valeur d'Event State. La valeur autorisée est BITSTRING-16,
- le deuxième donne la valeur d'Event Occurrence ID. La valeur autorisée est OCTETSTRING-2,
- le troisième donne la valeur d'Event Data. Les valeurs autorisées sont celles définies par Valeur de
Données (voir 5.2.3.1.12).--
CommonInformation ::= OCTETSTRING
--Le contenu d'OCTETSTRING est composé de la juxtaposition ordonnée des informations suivantes:

1- la valeur de la première information est de type BITSTRING-16 et représente «Event Type». Cette information obligatoire, indique le type d'objets event dont un événement est inclus dans la notification.

2- la valeur de la deuxième information est de type OCTETSTRING-2 et représente «EventTransactionOccurrenceID».

3- la valeur de la troisième information, facultative, est connue de l'utilisateur; elle occupe le reste des octets du contenu d'OCTETSTRING.--

```

EventType ::= BITSTRING-16 {
  common event detection support (0), --pris en charge si la valeur est 1.
  acknowledge action support (1), --pris en charge si la valeur est 1
  enable action support (2), --pris en charge si la valeur est 1
  C.S/User action support (3 à n) --à définir par les C.S/l'utilisateur
}

```

EventTransactionOccurrenceID ::= OCTETSTRING-2

--la sémantique est une initiative locale; elle doit être décrite dans la déclaration PICS.--

5.2.8.3 Définition du protocole du service confirmé acknowledge event notification

– Paramètres de demande

Ce service permet l'acquiescement de plusieurs objets événements. L'acquiescement d'un objet est effectué en mettant de force l'attribut de l'objet Ack State à la valeur: True (Vrai)..L'attribut «Ack State» est un composant de l'attribut «Event State».--

```

AcknowledgeEventNotificationRequest ::= SEQUENCE {
  list of event identification ListOfObjectID
  list of event occurrence ID type description SEQUENCE OF TypeDataSpecification
  --Le nombre de TypeDataSpecification peut être nul quand les types sont implicitement connus. La
  description de chaque TypeDataSpecification doit être de type OCTETSTRING-2. Étant donné que tous
  les descripteurs de type sont identiques, il est possible de réduire la liste à un seul descripteur--
  list of event occurrence ID value OCTETSTRING
  --Le contenu OCTET STRING est composé de la juxtaposition ordonnée des valeurs
  «EventOccurrenceID». Chacune de ces valeurs est de type OCTETSTRING-2--
}

```

– Paramètres de réponse

```

AcknowledgeEventNotificationResponse ::= SEQUENCE {
  list-of-access-result ListOfAccessResult,
  --ListOfAccessResult est de type BITSTRING. Le bit (0) est attribué au premier objet acquitté, le bit (1)
  est attribué au deuxième objet acquitté et ainsi de suite. Pour chaque objet acquitté, le bit qui lui est
  attribué indique si l'opération d'acquiescement a été exécutée avec succès ou s'est terminée par un
  échec. Le bit positionné à 0 signifie que l'acquiescement est un échec.
  list-of-ErrorType ListOfErrorType
  --le contenu de ListOfErrorType est vide si tous les acquiescements ont été effectués avec succès, sinon il
  contient la juxtaposition des valeurs ErrorType de tous les acquiescements qui se sont terminés par un
  échec. La première valeur correspond au premier échec, la deuxième valeur correspond au deuxième
  échec et ainsi de suite. Les valeurs autorisées Error Type sont celles définies en 5.2.3.1.10--
}

```

5.2.8.4 Définition du protocole du service confirmé alter event condition monitoring

– Paramètres de demande

Ce service permet de modifier les attributs d'Enable State ou d'Extension States de plusieurs objets event. Ces attributs sont des composants de l'attribut «Event State». La C.S ou l'utilisateur peuvent définir d'autres actions.

La modification d'un attribut «Event State» peut entraîner une nouvelle notification des événements du même objet. Le présent document n'indique ni comment ni quand cette nouvelle notification aura lieu--

```
AlterEventConditionMonitoringRequest ::= SEQUENCE {
    list of event state action ID      SEQUENCE OF EventStatePathAccess
    -- Ce paramètre indique la liste des attributs Event State à modifier. Pour chaque modification, elle
    -- indique l'identification de l'objet suivie par la ligne de l'action. Voir ci-dessous la description
    -- d'EventState avec les actions possibles--
    list of event state description    SEQUENCE OF TypeDataSpecification
    --Le nombre de TypeDataSpecification peut être nul quand les types sont implicitement connus. La
    --description de chaque «Event State» effectuée par TypeDataSpecification, doit être de type
    --OCTETSTRING-16. Lorsque tous les descripteurs de type sont identiques, il est possible de réduire la
    --liste à un seul descripteur--
    list of event state value          OCTETSTRING
    --Le contenu d'OCTET STRING est composé de la juxtaposition ordonnée des valeurs «EventState».
    --Chaque valeur Event State est représentée par une valeur de type BITSTRING-16 avec les contraintes
    --suivantes: les bits n'appartenant pas à l'action attendue doivent être mis de force à 0. Voir ci-dessous la
    --définition des actions d'«Event State»--
}
```

```
EventStatePathAccess ::= CHOICE {
    event-index/action-path-description [3] IMPLICIT SEQUENCE {
        event-index      UNSIGNED-16 (0..32K-1),
        action-rank      ActionRank },
    event-name/action-path-description CHOICE {
        undefined-length-name [4] IMPLICIT SEQUENCE {
            event-name      OCTETSTRING SIZE (0..32),
            action-rank      ActionRank },
        cs-defined-length-name [7] IMPLICIT SEQUENCE {
            event-name      OCTETSTRING SIZE (X),
            action-rank      ActionRank }
    }
}
```

NOTE CS-defined-length-name ne peut être utilisé que si les Normes d'Accompagnement ont défini la valeur X ≤ 32. Dans ce cas, la longueur d'OCTETSTRING SIZE (X), n'est plus codée. (Voir le codage dans l'Annexe 10.A).

```
ActionRank ::= UNSIGNED-16 {
    rank 0      (0), --valeur réservée
    rank 1      (1), --validation/invalidation
    rank 2 to n (2) à (n), --action utilisateur/CS
}
```

```
EventState ::= BITSTRING-16 {
    acknowledgement (0) --true si l'objet est acquitté
    validation      (1), --true si l'objet est validé
    extension-state (2) à (p), --l'utilisateur ou les CS peu(ven)t rallonger la liste des
    actions.
```

Dans ce but, nous recommandons d'une part de briser les bits de 2 à p en n ensembles ordonnés, chacun incluant 1 ou plusieurs bits et, d'autre part, d'affecter à chaque ensemble une action de rang 2 à n--

– Paramètres de réponse

```
AlterEventConditionMonitoringResponse ::= SEQUENCE {
    list-of-access-result      ListOfAccessResult,
    --ListOfAccessResult est du type BITSTRING. Le bit (0) est attribué au premier objet modifié, le bit (1)
    --est attribué au deuxième objet modifié et ainsi de suite. Pour chaque objet modifié, le bit qui lui est
    --attribué indique si l'opération de modification a été exécutée avec succès ou s'est terminée par un
    --échec. Un bit positionné à 0 signifie que la modification s'est terminée par un échec--
    list-of-ErrorType          ListOfErrorType
    --le contenu de ListOfErrorType est vide si toutes les modifications ont été exécutées avec succès,
    --sinon il contient la juxtaposition des valeurs ErrorType de toutes les modifications qui se sont terminées
    --par un échec. La première valeur doit correspondre au premier échec, la deuxième valeur doit
    --correspondre au deuxième échec et ainsi de suite. Les valeurs autorisées pour Error Type sont celles
    --définies en 5.2.3.1.10-
}
```

5.2.8.5 Définition du protocole du service confirmé get alarm summary

– Paramètres de demande.

```
GetAlarmSummaryRequest ::= CHOICE {
    without result events access [1] IMPLICIT ListOfObjectID ,
    --Ce choix indique une liste d'objets event dont l'attribut Event_Message doit être lu. Il indique
    --également que la réponse ne doit pas contenir des descripteurs du type de valeurs lues--
    with result events access [3] IMPLICIT ListOfObjectID
    --Ce choix indique une liste d'objets event dont l'attribut Event_Message doit être lu. Il indique
    --également que la réponse doit contenir les descripteurs du type de valeurs lues--
}
```

– Paramètres de réponse.

-- La procédure de lecture pour un attribut Event_Message exécuté avec succès entraîne le retour de sa valeur. Un échec de la procédure de lecture d'un attribut Event_Message entraîne le retour de la valeur ErrorType--

```
GetAlarmSummaryResponse ::= SEQUENCE {
    list-of-access-result      ListOfAccessResult,
    --ListOfAccessResult est du type BITSTRING. Le bit (0) est attribué au premier attribut lu, le bit (1) est
    --attribué au deuxième attribut lu et ainsi de suite. Pour chaque attribut lu, le bit qui lui est attribué indique
    --si l'opération de lecture a été exécutée avec succès ou s'est terminée par un échec. Un bit positionné à
    --0 signifie que la lecture est un échec--
    list-of-event-message-description SEQUENCE OF TypeDataSpecification --voir note 1--
    list-of-event-message-value      OCTETSTRING --voir note 2--
}
```

NOTE 1 Ce paramètre contient une liste de descripteurs. Cette liste pourrait être réduite à ZÉRO descripteur si la demande n'exige pas les types de descripteur. Chaque TypeDataSpecification décrit soit un type d'EventMessage si l'opération de lecture a été un succès, soit un Type d'Erreur si les opérations de lecture ont été un échec. Si toutes les opérations de lecture ont été réalisées avec succès et les descripteurs étaient du même type, la liste pourrait être réduite à un seul descripteur d'Event Message.

NOTE 2 Selon le type de l'objet, le descripteur «EventMessage» doit être celui d'une structure ayant:

a) soit uniquement un seul champ:

- ce champ décrit le type Event Data. Les types autorisés sont ceux définis par TypeDataSpecification à l'exception du type d'erreur.

b) soit deux champs:

- le premier champ décrit l'Event Occurrence ID. Le type autorisé est OCTETSTRING SIZE-2,
- le second champ décrit le Type Event Data. Les types autorisés sont ceux définis par TypeDataSpecification à l'exception du type d'erreur.

b) soit trois champs:

- le premier champ décrit le type Event State. Le type autorisé est BITSTRING-16,
- le premier champ décrit le type d'Event Occurrence ID. Le type autorisé est OCTETSTRING SIZE-2,
- le troisième décrit le type d'Event Data. Les types autorisés sont ceux décrits par TypeDataSpecification à l'exception du type d'erreur.--

Ce paramètre contient la juxtaposition des valeurs Event Message ou Error Type.

Chaque opération de lecture effectuée avec succès rétablit la valeur d'Event Message. Chaque opération de lecture qui s'est terminée par un échec rétablit la valeur d>Error Type.

NOTE 3 En fonction du type de l'objet, la valeur d'un «EventMessage» comprend:

a) soit uniquement un seul champ:

- ce champ donne la valeur d'Event Data. Les valeurs autorisées sont celles définies par Valeur de Données (voir 8.1.10.2).

b) soit deux champs:

- le premier champ donne la valeur d'Event Occurrence ID. La valeur autorisée est de type OCTETSTRING-2,
- le second donne la valeur d'Event Data. Les valeurs autorisées sont celles définies par Valeur de Données (voir 8.1.10.2),

b) soit trois champs:

- le premier champ donne la valeur d'Event State. La valeur autorisée est de type BITSTRING-16.

- le deuxième champ donne la valeur d'Event Occurrence ID. La valeur autorisée est de type OCTETSTRING-2,
- le troisième champ donne la valeur d'Event Data. Les valeurs autorisées sont celles définies par Valeur de Données (voir 8.1.10.2).--

5.2.8.6 Définition du protocole du service confirmé get event condition attribute

- Paramètres de demande.

GetEventConditionAttributesRequest ::= Identification

- Paramètres de réponse.

GetEventConditionAttributesResponse ::= SEQUENCE {
 list-of-identification SEQUENCE OF IDENTIFICATION
 object-class ObjectClass --valeur autorisée "Event"--
 event type EventType
 -- Ce paramètre indique le type d'objet. Pour la définition d'Event Type, voir le service Event Notification (voir 8.7.2)--
 event-message-type-description TypeDataSpecification
 -- Ce paramètre donne le descripteur d'«Event Message» d'un objet. La réponse du service "Get Alarm Summary" (voir 8.7.5) explique en détail comment construire une description d'Event Message--
 Event-access-protection ::= EventAccessProtection
 --Le contenu de ce paramètre est vide si l'objet n'est pas protégé--
}

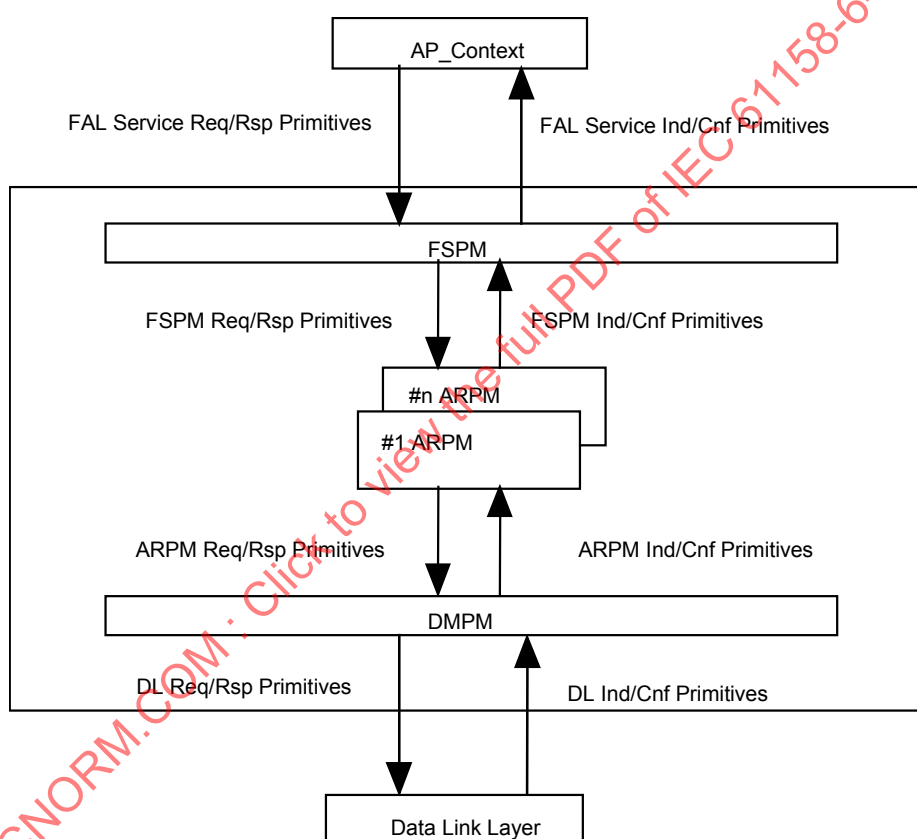
EventAccessProtection ::= BITSTRING {
 --Le contenu est vide si l'objet n'est pas protégé, sinon, il doit correspondre à la description suivante.
 --mot de passe--
 password bit 8 (0),
 password bit 7 (1),
 password bit 6 (2),
 password bit 5 (3),
 password bit 4 (4),
 password bit 3 (5),
 password bit 2 (6),
 password bit 1 (7),
 --groupes d'accès--
 access group 8 (8),
 access group 7 (9),
 access group 6 (10),
 access group 5 (11),
 access group 4 (12),
 access group 3 (13),
 access group 2 (14),
 access group 1 (15),
 --bit (16) réserve
 Dg (17),
 --si la valeur est 1 et si le client dispose des droits requis d'accès au groupe il peut valider/invalidier et lire l'objet à l'aide des services Alter Event Condition Monitoring et Get Alarm Summary--
 Wg (18),
 --si la valeur est 1 et si le client dispose des droits requis d'accès au groupe il peut délivrer un acquittement et lire l'objet à l'aide des services Acknowledge Event Notification et Get Alarm Summary.--
 --bits (19), (20) réservés
 D (21),
 --si la valeur est 1 et si le client fournit un mot de passe identique à celui spécifié dans l'attribut «password» , il peut valider/invalidier et lire l'objet à l'aide des services Alter Event Condition Monitoring et Get Alarm Summary--
 W (22),
 --si la valeur est 1 et si le client dispose des droits de mot de passe requis, il peut délivrer un acquittement et lire l'objet à l'aide des services Acknowledge Event Notification et Get Alarm Summary.--
 --bits (23), (24), (25), (26), (27), (28) sont réservés
 Da (29),
 --si la valeur est 1, tous les clients peuvent valider/invalidier et lire l'objet à l'aide des services Alter Event Condition Monitoring et Get Alarm Summary.--
 Wa (30),
 --si la valeur est 1, tous les clients peuvent délivrer un acquittement et lire l'objet à l'aide des services Acknowledge Event Notification et Get Alarm Summary.--
 --bit (31) réservé
 extension OCTETSTRING}

6 Structure des machines de protocole

Les interfaces avec les services de la FAL et les machines de protocole sont spécifiées dans cet article.

NOTE Les diagrammes d'états et les ARPM définis dans ce paragraphe définissent uniquement les événements valides pour chacun. Le traitement des événements invalides est une initiative locale.

Le comportement de la couche FAL est décrit par trois machines de protocole intégrées. Des ensembles spécifiques de ces machines de protocole sont définis pour les différents types d'AREP. Les trois machines de protocole sont: La Machine de Protocole des Services de la FAL (FSPM), la Machine de Protocole des Relations entre Applications (ARPM) et la Machine de Protocole de Mapping de la Couche de Liaison de Données (DMPM). La relation entre ces machines de protocole ainsi que les primitives échangées entre elles sont décrites dans la Figure 9.



Légende

Anglais	Français
Data Link Layer	Couche de Liaison de Données
DL Req/Rsp Primitives	Primitives Req/Rsp de DL
DL Ind/Cnf Primitives	Primitives Ind/Cnf de DL
ARPM Req/Rsp Primitives	Primitives Req/Rsp de l'ARPM
ARPM Ind/Cnf Primitives	Primitives Ind/Cnf de l'ARPM
FSPM Req/Rsp Primitives	Primitives Req/Rsp de la FSPM
FSPM Ind/Cnf Primitives	Primitives Ind/Cnf de la FSPM
FAL Service Req/Rsp Primitives	Primitives Req/Rsp des Services de la FAL
FAL Service Ind/Cnf Primitives	Primitives Ind/Cnf des Services de la FAL

Figure 9 – Relations entre les Machines de Protocole et les Couches Adjacentes

La FSPM décrit l'interface de service entre l'AP-Context et un AREP particulier. La FSPM est commune à toutes les classes AREP et ne présente aucun changement d'état. La machine de protocole FSPM est chargée des activités suivantes:

- a) accepter les primitives de service émises par l'utilisateur des services de la FAL et les convertir en primitives internes de la FAL;
- b) sélectionner un diagramme d'états de l'ARPM approprié en fonction du paramètre AREP Identifier fourni par l'AP-Context et envoyer des primitives internes de la FAL à l'ARPM sélectionnée;
- c) accepter les primitives internes de la FAL provenant de l'ARPM et les convertir en primitives de service destinées à l'AP-Context;
- d) émettre les primitives de service de la FAL à l'AP-Context basé sur le paramètre AREP Identifier associé aux primitives.

L'ARPM décrit l'établissement et la libération d'une AR et l'échange des PDU FAL avec une (des) ARPM distante(s). La machine de protocole ARPM est chargée des activités suivantes:

- a) accepter les primitives internes de la FAL provenant de la FSPM et créer et envoyer d'autres primitives internes de la FAL à la FSPM ou à la DMPM, basé sur les types d'AREP et de primitives;
- b) accepter les primitives internes de la FAL provenant de la DMPM et les envoyer à la FSPM comme une forme de primitives internes de la FAL;
- c) si les primitives sont destinées au service Establish ou Abort, il doit essayer d'établir ou de libérer l'AR spécifiée.

La DMPM décrit le mapping entre la FAL et la DLL. Elle est commune à tous les types d'AREP et ne présente aucun changement d'état. La DMPM est chargée des activités suivantes:

- a) accepter les primitives internes de la FAL provenant de l'ARPM, préparer les primitives des services de la DLL et les envoyer à la DLL;
- b) recevoir les primitives d'indication ou de confirmation de la DLL provenant de la DLL et les envoyer à l'ARPM sous la forme de primitives internes de la FAL.

Pour le protocole de la Couche d'Application de Type 7: les restrictions suivantes s'appliquent.

- Le diagramme d'états d'AP-Context n'est pas utilisé.
- Les diagrammes d'états de protocole des services de la FAL ne sont pas utilisés pour les ASE MPS mais sont utilisés globalement pour les sous-ensembles de la messagerie MMS (Sub-MMS) (ASE VMD, ASE Domain, ASE Program Invocation, ASE Variable, ASE Event, ASE Directory).
- La Machine de Protocole des Relations entre Applications (ARPM) et la Machine de Protocole de Mapping de la Couche de Liaison de Données (DMPM) sont combinées pour ASE AR MCS et ASE MPS:
- Une machine de protocole ARPM-DMPM pour ASE MPS,
- Une machine de protocole ARPM-DMPM pour ASE AR MCS.

7 Diagramme d'états d'AP-context

Caractéristique non utilisée

8 Machine de protocole des services de la FAL Sub-MMS (FSPM)

8.1 Généralités

Pour les besoins de la messagerie de bus de terrain, cette section impose la couche inférieure MCS définie dans la norme.

8.2 Projection des PDU SUB-MMS sur les services de MCS

SUB-MMSpdu	Service de couche inférieure MCS
Confirmed Request PDU	(A-Data ou A-Unidata) avec/sans acquittement.
Confirmed Response PDU	(A-Data ou A-Unidata) avec/sans acquittement.
Confirmed Error PDU	(A-Data ou A-Unidata) avec/sans acquittement.
Unconfirmed Request PDU	(A-Data ou A-Unidata) avec/sans acquittement.
Reject PDU	(A-Data ou A-Unidata) avec/sans acquittement.
Initiate Request PDU	A-Associate.Request,
Initiate Response PDU	A-Associate.Response(result +),
Initiate Error PDU	A-Associate.Response(result -),
Conclude Request PDU	A-Release.Request,
Conclude Response PDU	A-Release.Response(result +),
Conclude Error PDU	A-Release.Response(result -).

8.3 Projection du service abort de SUB-MMS sur les services de MCS

Le service Abort de SUB-MMS sans PDU doit donc être directement projeté sur le service A-Abort de MCS; par conséquent, les actions suivantes seront assurées par le fournisseur du service de SUB-MMS:

- à la réception d'une demande de service Abort (générée par l'utilisateur de Sub-MMS), il doit générer une demande A-Abort vers la couche inférieure,
- à la réception d'une indication de service A-Abort (générée par la couche inférieure) il doit générer une indication Abort vers l'UTILISATEUR.

8.4 Construction d'une PDU-SUB-MMS à partir d'une primitive de service

Dès réception d'une primitive de service (demande/réponse) autre que «Abort», le fournisseur de service SUB-MMS doit:

- construire une PDU conformément au protocole SUB-MMS,
- fournir la PDU construite à la couche inférieure par l'intermédiaire d'un service MCS conformément aux exigences de projection de PDU Sub-MMS pour les services de MCS.

8.5 Extraction d'une primitive de service valide à partir d'une PDU SUB-MMS

Dès réception d'une primitive de service MCS autre que A_Abort les actions suivantes doivent être effectuées dans l'ordre indiqué par le fournisseur de service SUB-MMS:

- vérifier que le paramètre "user-data" de la primitive contient une PDU valide (conformément au protocole SUB-MMS);
- si la PDU reçue est valide, extraire une primitive de service SUB-MMS et l'envoyer à l'utilisateur Sub-MMS;
- si la PDU reçue est invalide et le transfert a été effectué au moyen d'une association établie (réception réussie de «InitiateReqPDU» et échec de l'émission de «InitiateRespPDU») alors les deux actions suivantes peuvent être effectuées:
 - générer une indication Abort et l'émettre à la Sub-MMS,
 - générer une demande «A-Abort» au MCS de sorte que celui-ci libère les ressources réservées. Cette demande ne sera pas transférée au correspondant distant étant donné que le MCS en est incapable (Voir ASE MCS);

- d) si la PDU n'est pas valide et si le transfert de la PDU a été réalisé au moyen d'une association négociée, d'une association prédéfinie ou sans association, alors la CEI 61158-5-7, doit être respectée pour exécuter les deux actions suivantes:
- 1) construire une RejectPDU et l'émettre à la couche conformément aux exigences de projection de la PDU Sub-MMS pour les services de MCS,
 - 2) générer une indication du service Reject et l'émettre à l'utilisateur de Sub-MMS.

8.6 Négociation d'une syntaxe abstraite et d'une syntaxe de transfert généralement appelée contexte de présentation

Ce paragraphe définit la négociation dans le cadre suivant: la couche inférieure est MCS. Il ne montre pas les réductions opérées par les différents intervenants lors de la négociation. Ces réductions sont expliquées en détail dans un service Initiate de Sub-MMS.

Révision: la norme MCS autorise une seule syntaxe abstraite et une seule syntaxe de transfert par association.

Séquencement d'une ouverture d'association

1 – Utilisateur appelant:

L'utilisateur de Sub-MMS qui souhaite ouvrir une association avec un partenaire distant génère au fournisseur de service Sub-MMS une primitive Initiate.Request.

2 - Fournisseur de service de Sub-MMS de l'utilisateur appelant:

Dès réception de la primitive Initiate.Request initiée par l'utilisateur local Sub-MMS, le fournisseur de service Sub-MMS réserve les ressources nécessaires, construit une Initiate.Request PDU et génère vers la couche MCS le service A_Associate.Request avec le paramètre User Data = Initiate.Request PDU.

3 - Fournisseur de service de Sub-MMS de l'utilisateur appelé:

Dès réception d'un service A_Associate.Indication initié par la couche MCS, la couche Sub-MMS effectue les actions suivantes, selon les besoins:

- a) si elle note que le paramètre User Data du service A_Associate ne contient pas de Initiate.Request PDU valide ou si elle est incapable d'accepter l'ouverture d'une association pour diverses raisons, elle génère une primitive A_Associate.Response à la couche MCS avec les paramètres User Data = Empty et Result = negative à cause de l'utilisateur (l'utilisateur est le fournisseur de service de Sub-MMS).
- b) si elle note que le paramètre User Data du service A_Associate contient une Initiate.Request PDU valide et accepte l'ouverture d'une association, alors elle réserve les ressources nécessaires et génère la primitive Initiate.Indication à l'utilisateur Sub-MMS.

4 – Utilisateur appelé:

L'utilisateur Sub-MMS peut accepter ou refuser l'association.

1) Acceptation de l'association:

Suite à la réception de la primitive Initiate.Indication l'utilisateur de Sub-MMS génère vers le fournisseur de service Sub-MMS une primitive Initiate.Response (+) et prend des dispositions pour le futur échange de messages par le biais de cette voie de communication.

2) Refus de l'association:

Suite à la réception de la primitive Initiate.Indication l'utilisateur de Sub-MMS génère vers le fournisseur de service Sub-MMS une primitive Initiate.Response(-) et considère l'association proposée comme inexistante.

5 - Fournisseur de service de Sub-MMS de l'utilisateur appelé:

Le comportement de la couche varie en fonction de la réponse de l'utilisateur de Sub-MMS.

- 1) Dès réception d'une primitive Initiate.Response(+), le fournisseur de service Sub-MMS construit une Initiate.Response PDU et génère vers la couche MCS le service A_Associate.Response avec les paramètres User Data = Initiate.ResponsePDU et Results = positive et prend des dispositions pour les futurs échanges de messages par le biais de cette voie de communication.
- 2) Dès réception d'une primitive Initiate.Response(-), le fournisseur de service Sub-MMS libère les ressources réservées, construit une Initiate.Error PDU et génère vers la couche MCS le service A_Associate.Response avec les paramètres User Data = Initiate.ErrorPDU et Result = negative en raison de l'utilisateur et considère l'association proposée comme inexistante.

6 - Fournisseur de service de Sub-MMS de l'utilisateur appelant:

Le comportement du fournisseur de service Sub-MMS varie en fonction de la réponse reçue.

- 1) Dès réception d'une primitive A_Associate.Confirm(+), le fournisseur de service Sub-MMS effectue les actions suivantes, selon les besoins:
 - a) s'il note que le paramètre User Data du service A_Associate ne contient pas de Initiate.ResponsePDU valide, il ignore le service.
 - b) s'il note que le paramètre user data du service A_Associate contient une Initiate.Response PDU valide, il génère la primitive Initiate.Confirm(+) vers l'utilisateur de Sub-MMS et prend des dispositions pour les futurs échanges de messages par le biais de cette voie de communication.
- 2) Dès réception d'une primitive A_Associate.Confirm(-), le fournisseur de service Sub-MMS effectue les actions suivantes, selon les besoins:
 - a) s'il note que le paramètre user data du service A_Associate ne contient pas d'Initiate.ErrorPDU valide, il ignore le service.
 - b) s'il note que le paramètre user data du service A_Associate contient une Initiate.ErrorPDU valide il génère la primitive Initiate.Confirm(-) vers l'utilisateur de Sub-MMS, libère les ressources réservées et considère l'association proposée comme inexistante.

7 – Utilisateur appelé:

Le comportement de l'utilisateur de Sub-MMS varie en fonction de la réponse reçue.

- 1) Réponse positive:

Suite à la réception de la primitive Initiate.Confirm(+), l'utilisateur de Sub-MMS prend ses dispositions pour les futurs échanges de messages par le biais de cette voie de communication.
- 2) Réponse négative:

Suite à la réception de la primitive Initiate.Confirm(-), l'utilisateur de Sub-MMS considère l'association proposée comme inexistante.

NOTE 1 S'il est nécessaire de vérifier l'état d'une association négociée ou prédéfinie, en l'absence de trafic, le fournisseur de service Sub-MMS peut générer périodiquement un service de demande de MCS A_Data avec le paramètre User data = empty. La mise en œuvre de ce mécanisme du client relève d'une initiative locale.

NOTE 2 S'il est nécessaire de vérifier l'état d'une association négociée ou prédéfinie, en l'absence de trafic, le fournisseur de service Sub-MMS peut demander la réception périodique du service Sub-MMS A_Data request avec le paramètre User data = empty.

8.7 Identification de la syntaxe abstraite de SUB-MMS core

- 1) La syntaxe abstraite du module CORE-SUB-MMS est identifiée par un identifiant d'objet et un descripteur d'objet. Cette identification caractérise la syntaxe abstraite du Sub-MMS core.

Le présent document leur attribue les valeurs suivantes:

Valeur de l'identifiant d'objet selon ASN.1 DES BUS DE TERRAIN:
 {iso standard 9506 part-protocol (2)
 SUB-MMS-abstract-syntax-version1 (4) }

Valeur du descripteur d'objet selon ASN.1 DES BUS DE TERRAIN:
 «sub-mms-abstract-syntax-major-version1».

- 2) Le SUB-MMS-General-module-1 est identifié par un identifiant d'objet et un descripteur d'objet. Cette identification caractérise le type d'ASE Sub-MMS car il permet de générer différentes syntaxes abstraites.

Le présent document leur attribue les valeurs suivantes:

Valeur de l'identifiant d'objet selon ASN.1 DES BUS DE TERRAIN:
 {iso standard 9506 part_protocol (2)
 sub-mms-general-module-version1 (5) }

Valeur du descripteur d'objet selon ASN.1 DES BUS DE TERRAIN:
 «sub-mms-general-module-major-version1»

- 3-La syntaxe de transfert relative à la syntaxe abstraite du CORE-SUB-MMS et de la CS est identifiée par un identifiant d'objet et un descripteur d'objet:

Valeur de l'identifiant d'objet selon ASN.1 DES BUS DE TERRAIN:
 {iso standard 8825 fieldbus-encoding (5) }.

Valeur du descripteur d'objet selon ASN.1 DES BUS DE TERRAIN:
 «fieldbus-encoding-rules»,

La version majeure de ce document est 1.

La version mineure de ce document est 1.

Ce document autorise d'autres syntaxes de transfert.

8.8 Identification du nom de contexte d'application

Pour l'utilisation spécifique de la couche d'application, le présent document propose la couche d'application contenant un «CORE-SUB-MMS», comme l'ASE identifié par un identifiant d'objet et un descripteur d'objet.

Le présent document leur attribue les valeurs suivantes:

Valeur de l'identifiant d'objet selon ASN.1 DES BUS DE TERRAIN:
 {iso standard 9506 part-protocol (2)
 sub-mms-application-context-version1 (6)}

Valeur du descripteur d'objet selon ASN.1 DES BUS DE TERRAIN:
 «iso sub-mms».

8.9 Identification de l'ASE de la syntaxe abstraite centrale et de la syntaxe de transfert

Ce paragraphe définit les équivalences suivantes pour le MCS:

- 1- Pour l'identifiant d'objet suivant:

Valeur de l'identifiant d'objet selon ASN.1 DES BUS DE TERRAIN (modèle général Sub-MMS):

{iso standard 9506 part_protocol (2)
sub-mms-general-module-version1 (5) }

nous attribuons la valeur binaire {0} au paramètre «TYPE of ASE» du service A_Associate de MCS.

2 - Pour l'identifiant d'objet suivant (Sub-MMS core):

FIELD BUS ASN.1 object-identifier value:
{iso standard 9506 part_protocol (2)
SUB-MMS-abstract-syntax-version1 (4) }

nous attribuons la valeur binaire {0} au paramètre "ABSTRACT SYNTAX" du service A_Associate de MCS.

3 - Pour l'identifiant d'objet suivant (règle de codage Sub-MMS):

Valeur de l'identifiant d'objet selon ASN.1 DES BUS DE TERRAIN:
{iso standard 8825 fieldbus-encoding (5) }.

nous attribuons la valeur binaire {0} au paramètre «TRANSFERT SYNTAX» du service A_Associate.

9 Machine de protocole des relations entre associations (ARPM)

Ces machines sont définies dans cette partie de la norme avec la Machine de protocole de mapping de la DLL, voir le chapitre suivant.

10 Machine de protocole de mapping de la DLL (DMPM)

10.1 ARPM et DMPM de MPS

10.1.1 Généralités

Les procédures du protocole spécifient:

- les interactions entre les services d'application MPS et les services de couche de liaison;
- l'interaction entre les éléments des services d'application MPS et les services de couche de liaison. Les éléments de service sont caractérisés par des mécanismes internes de la couche application qui ne peuvent pas être localement accessibles par les primitives offertes à l'utilisateur.

Le mapping entre les paramètres spécifiant les défauts de liaison et les paramètres *Type of Error* des primitives du service de la couche application n'est pas spécifié. Les types d'erreur décrits sont simplement des recommandations.

10.1.2 Procédures de service

10.1.2.1 Généralités

Les procédures de protocole des services d'application décrivent les interactions avec les services de liaisons au moyen des réseaux d'évaluation.

Cependant, le comportement décrit par les réseaux d'évaluation ne présume pas de la mise en œuvre interne des services des diagrammes d'états finis de la MPS.

En outre, les procédures de service décrivent le mapping sur les services de liaison en termes de primitives et de paramètres.

10.1.2.2 Service de lecture locale

10.1.2.2.1 Description du service

Le service de lecture locale interagit uniquement avec les services de liaison lors de la lecture d'une variable non resynchronisée. Lorsqu'une variable est resynchronisée le service effectue une lecture sur les attributs *Private Value* et *Private Status* de l'objet d'application *Consumed Variable Local Image*, situé au niveau de la couche application.

10.1.2.2.2 Réseau d'évaluation

Le réseau d'évaluation du service A_Readloc est représenté à la Figure 10.

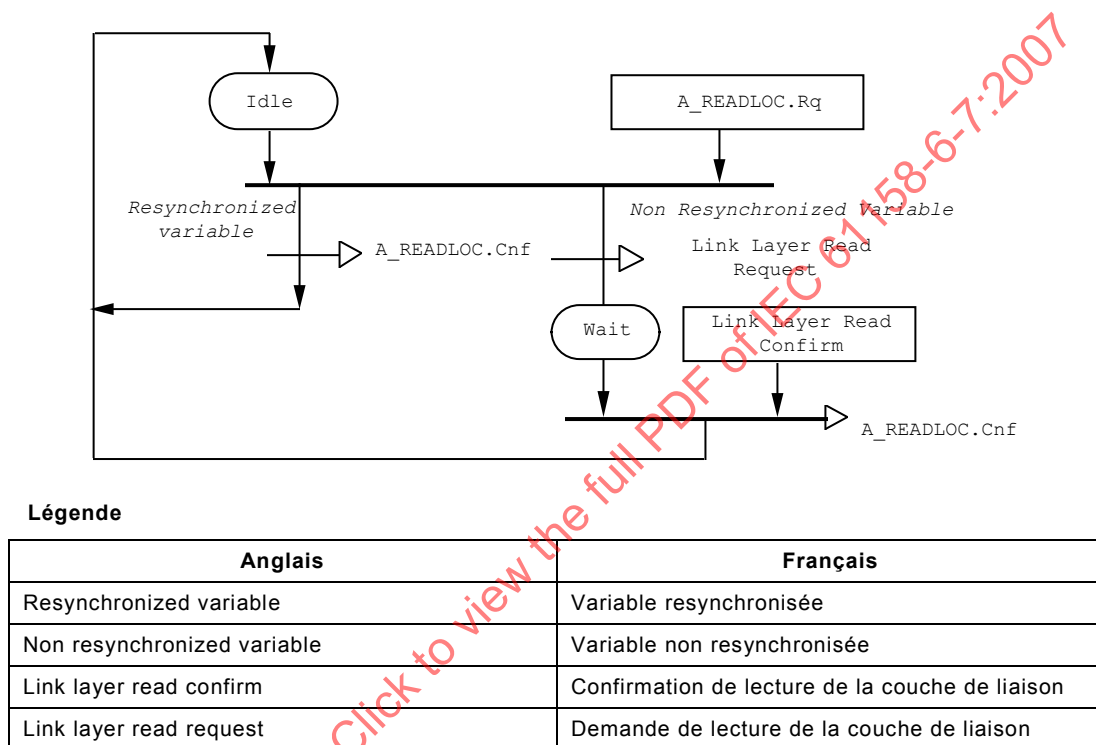


Figure 10 – Réseau d'évaluation du service A_Readloc

La confirmation associée à une demande de lecture A_READLOC.rq doit être négative lorsque la confirmation de liaison de données est elle-même négative.

10.1.2.2.3 Mapping sur les services de liaison

Le mapping sur les services de la couche de liaison est comme suit:

Demande de Lecture de la Couche de Liaison: L_GET.request(IDENTIFIER)

Confirmation de Lecture de la Couche de Liaison: L_GET.confirm(IDENTIFIER,STATUS, L_DATA)

10.1.2.2.4 Mapping sur les paramètres de liaison

Le mapping des attributs des objets MPS sur les paramètres des services de liaison est effectué comme suit:

IDENTIFIER: Ce paramètre correspond à l'attribut *Identifieur* de l'objet *Consumed Variable Local Image*.

L_DATA: Ce paramètre correspond à la PDU de syntaxe de transfert résultant des règles de codage. Les composants du contenu de la PDU sont disponibles dans les attributs *Public value* et *Transmitted status value* de l'objet *Consumed Variable Local Image*.

STATUS: Lorsque la confirmation est négative, ce paramètre spécifie le type d'erreur.

10.1.2.3 Service d'écriture locale

10.1.2.3.1 Description du service

Le service d'écriture locale interagit uniquement avec les services de liaison lors de l'écriture d'une variable non resynchronisée. Lorsqu'une variable est resynchronisée, le service écrit la valeur des attributs *Private Value* de l'objet d'application *Produced Variable Local Image*, situé au niveau de la couche application.

10.1.2.3.2 Réseau d'évaluation

Le réseau d'évaluation du service A_Writeloc est représenté à la Figure 11.

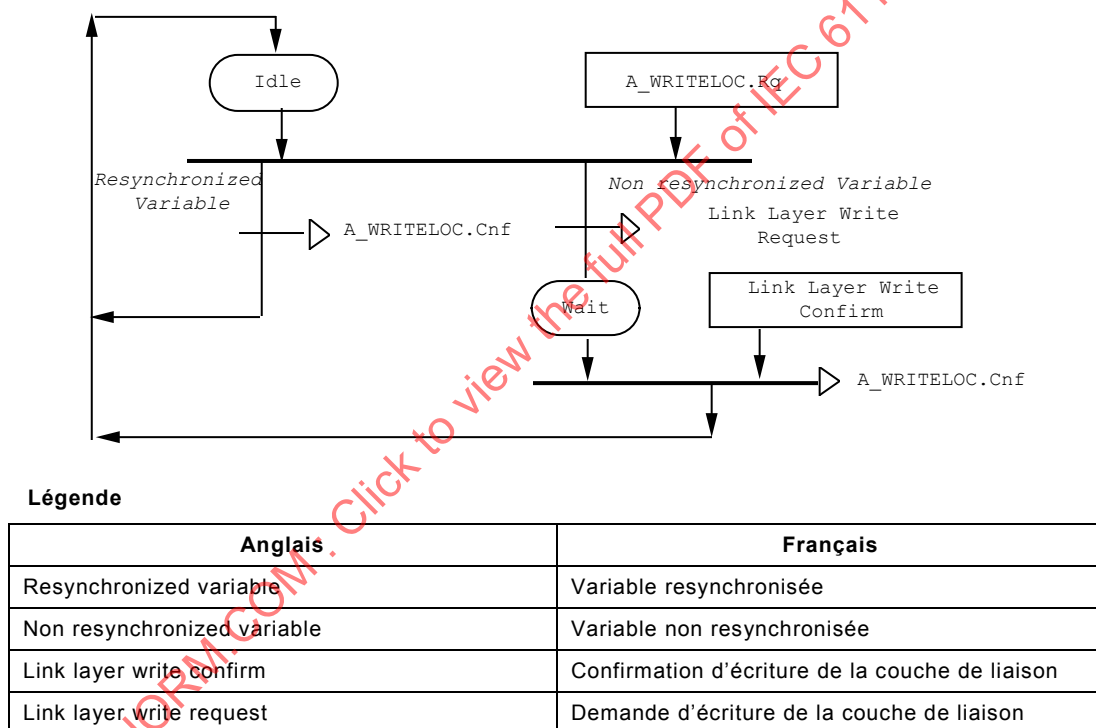


Figure 11 – Réseau d'évaluation du service A_Writeloc

La confirmation associée à une demande d'écriture A_WRITELOC.rq doit être négative lorsque la confirmation de liaison de données est elle-même négative.

10.1.2.3.3 Mapping sur les services de liaison

Le mapping sur les services de la couche de liaison est comme suit:

Demande d'Écriture de la Couche de Liaison: L_PUT.request(IDENTIFIER, L_DATA)

Confirmation d'Écriture de la Couche de Liaison: L_PUT.confirm(IDENTIFIER, STATUS)

10.1.2.3.4 Mapping sur les paramètres de liaison

Le mapping des attributs des objets MPS sur les paramètres des services de liaison est effectué comme suit:

IDENTIFIER: Ce paramètre correspond à l'attribut *Identifier* de l'objet *Produced Variable Local Image*.

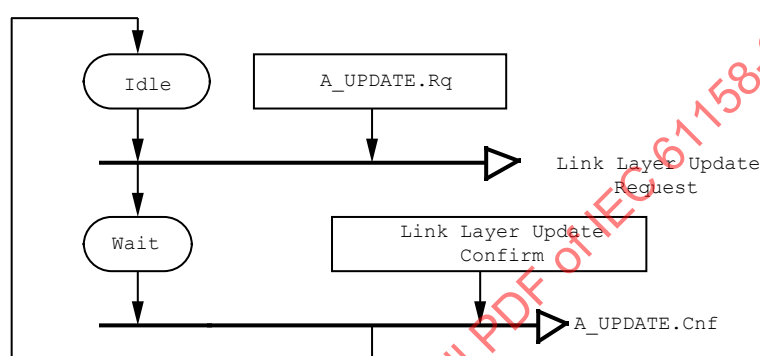
L_DATA: Ce paramètre correspond à la PDU de syntaxe de transfert résultant des règles de codage. Les composants du contenu de la PDU sont disponibles dans les attributs *Public value* et *Transmitted status value* de l'objet *Produced Variable Local Image*.

STATUS: Lorsque la confirmation est négative, ce paramètre spécifie le type d'erreur.

10.1.2.4 Service de mise à jour

10.1.2.4.1 Réseau d'évaluation

Le réseau d'évaluation du service *A_Update* est représenté à la Figure 12.



Légende

Anglais	Français
Link layer update confirm	Confirmation de mise à jour de la couche de liaison
Link layer update request	Demande de mise à jour de la couche de liaison

Figure 12 – Réseau d'évaluation du service *A_Update*

La confirmation associée à une demande de mise à jour *A_UPDATE.rq* doit être négative lorsque la confirmation de liaison de données est elle-même négative.

10.1.2.4.2 Mapping sur les services de liaison

Le mapping sur les services de la couche de liaison de données est effectué comme suit:

Demande de Mise à Jour de la Couche de Liaison:

L_FREE_UPDATE.request(SEQUENCE OF REQUESTED IDENTIFIERS, PRIORITY)

Confirmation de Mise à Jour de la Couche de Liaison:

L_FREE_UPDATE.confirm(STATUS)

10.1.2.4.3 Mapping sur les paramètres de liaison

Le mapping des attributs des objets MPS sur les paramètres des services de liaison est effectué comme suit:

SEQUENCE OF REQUESTED IDENTIFIERS: Ce paramètre correspond à l'attribut *Identifier* de l'un des objets suivants: Objet *Consumed Variable Local Image*, *Produced Variable Local Image* ou *Third Part Variable Local Image*. Cet attribut correspond à la variable spécifiée dans le paramètre *Variable Specification* de la primitive de service *A_UPDATE*.

PRIORITY: Ce paramètre correspond au paramètre *Priority* de la primitive de service A_UPDATE.

STATUS: Lorsque la confirmation est négative, ce paramètre spécifie le type d'erreur.

10.1.2.5 Service de lecture à distance

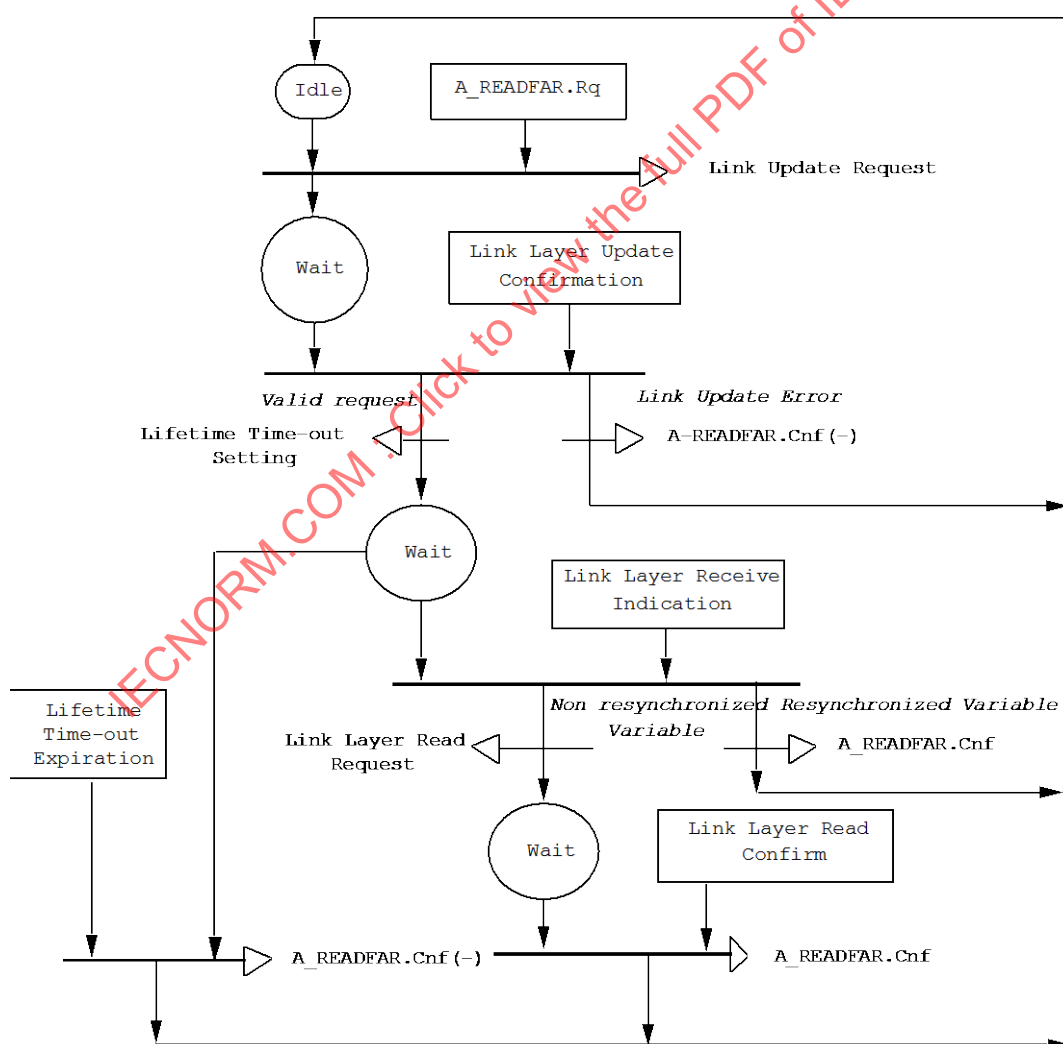
10.1.2.5.1 Description du service

Lorsqu'une variable est resynchronisée, le service de lecture à distance effectue uniquement une demande de mise à jour de liaison. La valeur lue étant l'attribut de *Private value* de l'objet *Consumed Variable Local Image*.

Lorsqu'une variable n'est pas resynchronisée le service de lecture à distance effectue d'abord une demande de mise à jour de liaison et, après confirmation de cette demande, le service effectue une lecture locale sur l'attribut *Public value* de l'objet *Consumed Variable Local Image*.

10.1.2.5.2 Réseau d'évaluation

Le réseau d'évaluation du service A_Readfar est représenté à la Figure 13.



Légende

Anglais	Français
Link Update request	Demande de Mise à jour de la Liaison
Link Layer Update Confirmation	Confirmation de Mise à jour de la Couche de Liaison
Valid request	Demande valide
Lifetime Time-out Setting	Réglage de la Temporisation de la durée de vie
Lifetime Time-out Expiration	Expiration de la Temporisation de la durée de vie
Link Update Error	Erreur de Mise à jour de la Liaison
Link Layer Read Request	Demande de Lecture de la Couche de Liaison
Non resynchronized variable	Variable non resynchronisée
Resynchronized variable	Variable resynchronisée
Link Layer Read Confirmation	Confirmation de Lecture de la Couche de Liaison

Figure 13 – Réseau d'évaluation du service A_Readfar

10.1.2.5.3 Mapping sur les services de Liaison

Le mapping sur la Couche de Liaison de données est effectué comme suit:

Demande de Lecture de la Couche de Liaison: L_GET.(IDENTIFIER)

Confirmation de Lecture de la Couche de Liaison: L_GET.confirm(IDENTIFIER,STATUS,
L_DATA)

Demande de Mise à Jour de la Couche de Liaison: L_FREE_UPDATE.request(SEQUENCE
OF REQUESTED IDENTIFIERS, PRIORITY)

Confirmation de Mise à Jour de la Couche de Liaison:
L_FREE_UPDATE.confirm(STATUS)

Indication de Réception de la Couche de Liaison: L_RECEIVED.indication(IDENTIFIER)

10.1.2.5.4 Mapping sur les paramètres de liaison

Le mapping des attributs des objets MPS sur les paramètres des services de liaison est effectué comme suit:

IDENTIFIER: Ce paramètre correspond à l'attribut *Identifieur* de l'objet *Consumed Variable Local Image*.

L_DATA: Ce paramètre correspond à la PDU de syntaxe de transfert résultant des règles de codage. Les composants du contenu de la PDU sont disponibles dans les attributs *Public value* et *Transmitted status value* de l'objet *Consumed Variable Local Image*.

SEQUENCE OF REQUESTED IDENTIFIERS: Ce paramètre correspond à l'attribut *Identifieur* de l'objet *Consumed Variable Local Image*.

PRIORITY: Ce paramètre correspond au paramètre *Priority* de la primitive de service A_READFAR.

STATUS: Lorsque la primitive de confirmation est négative, ce paramètre spécifie le type d'erreur.

10.1.2.6 Service d'écriture à distance

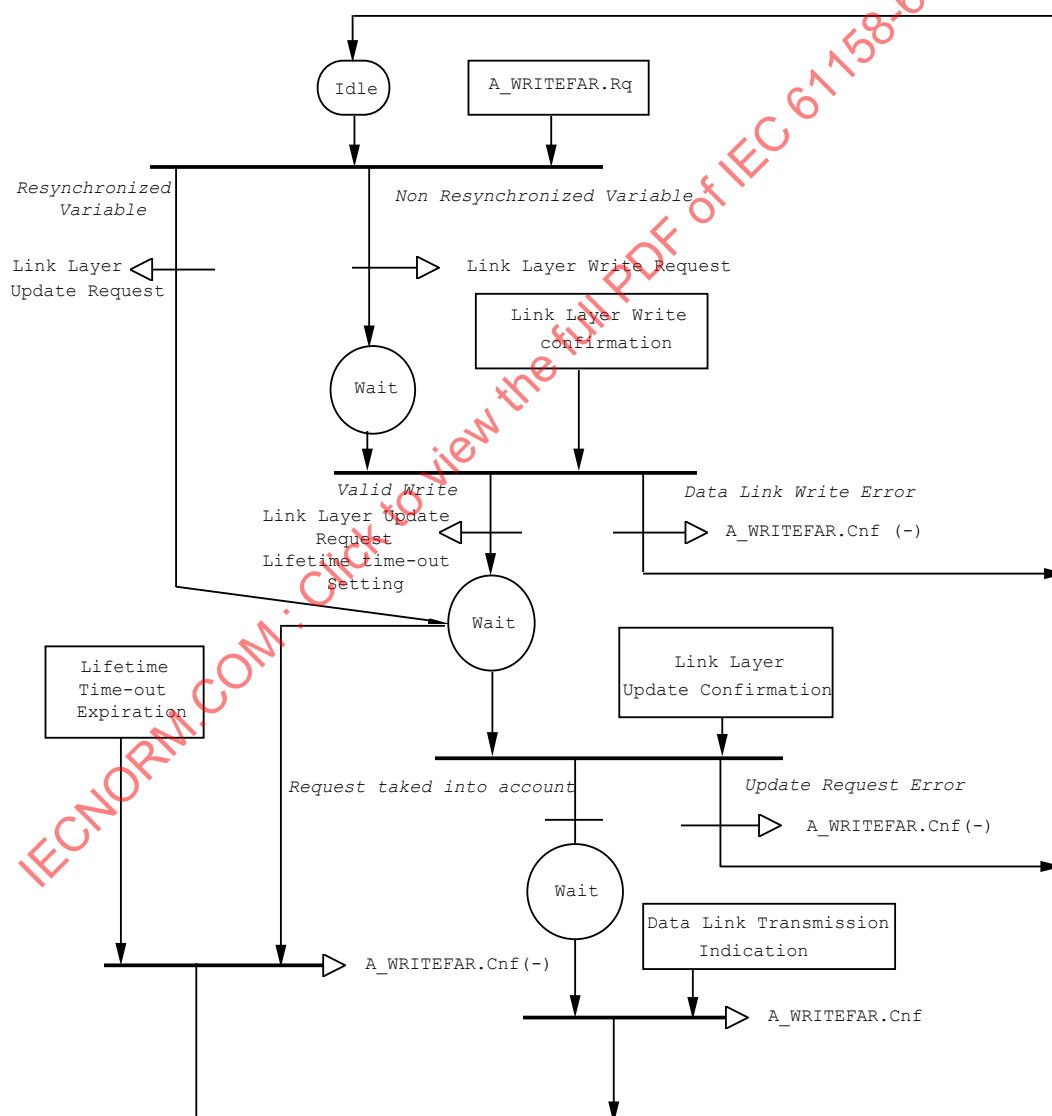
10.1.2.6.1 Description du service

Lorsqu'une variable est resynchronisée, le service d'écriture à distance effectue uniquement une demande de mise à jour de liaison, la valeur de la variable ayant déjà été écrite sur l'attribut *Private value* de l'objet *Produced Variable Local Image*.

Lorsqu'une variable n'est pas resynchronisée le service d'écriture à distance effectue d'abord une écriture locale sur l'attribut *Public value* de l'objet *Produced Variable Local Image*, et, après confirmation de l'écriture locale, le service effectue une demande de mise à jour de liaison.

10.1.2.6.2 Réseau d'évaluation

Le réseau d'évaluation du service A_writefar est représenté à la Figure 14.



Légende

Anglais	Français
Resynchronized variable	Variable resynchronisée
Link Layer Update request	Demande de Mise à Jour de la Couche de Liaison

Anglais	Français
Non Resynchronized Variable	Variable Non Resynchronisée
Link Layer Write Request	Demande d'Écriture de la Couche de Liaison
Link Layer Write Confirmation	Confirmation d'Écriture de la Couche de Liaison
Valid Write	Écriture valide
Lifetime Time-out Setting	Réglage de la Temporisation de la durée de vie
Data Link Write Error	Erreur d'Écriture de la Liaison de Données
Link Layer Update Confirmation	Confirmation de Mise à Jour de la Couche de Liaison
Request taked into account	Demande prise en compte
Update Request Error	Erreur de Demande de Mise à Jour
Data Link Transmission Indication	Indication d'émission de Liaison de Données
Lifetime Time-out Expiration	Expiration de la Temporisation de la durée de vie

Figure 14 – Réseau d'évaluation du service A_Writefar

10.1.2.6.3 Mapping sur les services de liaison

Le mapping sur la couche de Liaison de Données est effectué comme suit:

Demande d'Écriture de la Couche de Liaison: L_PUT.request(IDENTIFIER, L_DATA)
 Confirmation d'Écriture de la Couche de Liaison: L_PUT.confirm(IDENTIFIER, STATUS)
 Demande de Mise à Jour de la Couche de Liaison: REQUESTED IDENTIFIERS, PRIORITY)
 Confirmation de Mise à Jour de la Couche de Liaison: L_FREE_UPDATE.confirm(STATUS)
 Indication d'Émission de la Couche de Liaison: L_SENT.indication(IDENTIFIER)

10.1.2.6.4 Mapping sur les paramètres de liaison

Le mapping des attributs des objets MPS sur les paramètres des services de liaison est effectué comme suit:

IDENTIFIER: Ce paramètre correspond à l'attribut *Identifieur* de l'objet *Produced Variable Local Image*

L_DATA: Ce paramètre correspond à la PDU de syntaxe de transfert résultant des règles de codage. Les composants du contenu de la PDU sont disponibles dans les attributs *Public value* et *Transmitted status value* de l'objet *Produced Variable Local Image*.

SEQUENCE OF REQUESTED IDENTIFIERS: Ce paramètre correspond à l'attribut *Identifieur* de l'objet *Produced Variable Local Image*.

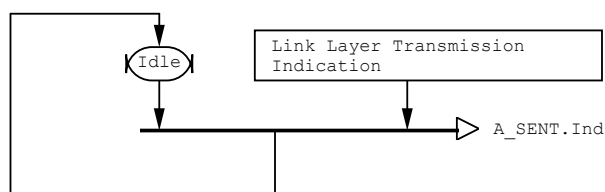
PRIORITY: Ce paramètre correspond au paramètre *Priority* de la primitive de service A_WRITEFAR.

STATUS Lorsque la primitive de confirmation est négative, ce paramètre spécifie le type d'erreur.

10.1.2.7 Service d'Indication d'Émission

10.1.2.7.1 Réseau d'évaluation

Le réseau d'évaluation du service A_Sent est représenté à la Figure 15.

**Légende**

Anglais	Français
Link Layer Transmission Indication	Indication d'Émission de la Couche de Liaison

Figure 15 – Réseau d'évaluation du service A_Sent**10.1.2.7.2 Mapping sur les services de liaison**

Le mapping sur les services de Couche de Liaison est effectué comme suit:

Indication d'Émission de la Couche de Liaison: L_SENT.indication(IDENTIFIER)

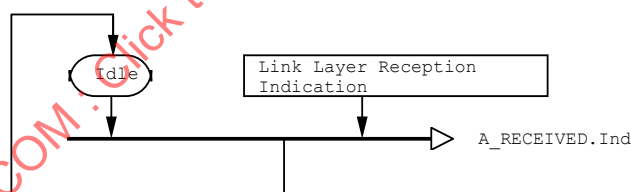
10.1.2.7.3 Mapping sur les paramètres de liaison

Le mapping des attributs des objets MPS sur les paramètres des services de liaison est effectué comme suit:

IDENTIFIER: Ce paramètre correspond à l'attribut *Identifier* de l'objet *Produced Variable Local Image* de la variable qui met en œuvre le service d'indication d'émission.

10.1.2.8 Service d'indication de réception**10.1.2.8.1 Réseau d'évaluation**

Le réseau d'évaluation du service A_Received est représenté à la Figure 16.

**Légende**

Anglais	Français
Link Layer Reception Indication	Indication de Réception de la Couche de Liaison

Figure 16 – Réseau d'évaluation du service A_Received**10.1.2.8.2 Mapping sur les services de liaison**

Le mapping sur les services de la Couche de Liaison est effectué comme suit:

Indication de Réception de la Couche de Liaison: L_RECEIVED.indication(IDENTIFIER)

10.1.2.8.3 Mapping sur les paramètres de liaison

Le mapping des attributs des objets MPS sur les paramètres des services de liaison est effectué comme suit:

IDENTIFIER: Ce paramètre correspond à l'attribut *Identifier* de l'objet *Consumed Variable Local Image* de la variable qui met en œuvre le service d'indication d'émission.

10.1.3 Procédures des éléments de service

10.1.3.1.1 Généralités

Les éléments des services d'application sont définis par des mécanismes internes de la Couche d'Application qui ne peuvent pas être accessibles par des primitives d'application dédiées.

Les procédures du protocole associées aux éléments des services décrivent l'interaction avec les services de liaison à l'aide des réseaux d'évaluation; de plus, les procédures de service décrivent le mapping de ces mécanismes sur les services de liaison du AL TYPE 7environment en termes de primitives et de paramètres.

10.1.3.2 Mapping des demandes et des actions des mécanismes d'élaboration du statut de validité de production:

Un objet *Produced Variable Local Image* possède une procédure d'éléments de service associée au statut de validité de production, la variable est soit resynchronisée sur la production, soit non resynchronisée.

Mise à jour du statut:

Cette action est mappée sur la primitive de service de niveau de liaison de données

L_PUT.request(IDENTIFIER,L_DATA)

Le paramètre IDENTIFIER correspond à la valeur de l'attribut *Identifier* de l'objet *Produced Variable Local Image*.

Le paramètre L_DATA correspond aux valeurs des attributs *Public Value* et *Transmitted Status Value* de l'objet *Produced Variable Local Image*.

Ordre de synchronisation:

Cette demande est mappée sur une primitive de service de niveau de la liaison de données

L_RECEIVED.indication(IDENTIFIER) or

L_SENT.indication(IDENTIFIER)

Le paramètre IDENTIFIER, correspond à la valeur de l'attribut *Identifier* de l'objet *Consumed Variable Local Image* (de synchronisation) ou de l'objet *Produced Variable Local Image* (de synchronisation) associé à la variable de synchronisation impliquée dans l'élaboration du statut de validité de production.

Lorsque les deux objets *Consumed Variable Local Image* et *Produced Variable Local Image* associés à la variable de synchronisation résident dans la même entité d'application, l'ordre de synchronisation est mappé sur la réception de la variable consommée.

10.1.3.3 Mapping des demandes et des actions des mécanismes d'élaboration du statut de validité d'émission

Un objet *Consumed Variable Local Image* possède une procédure des éléments de service associée au statut de validité d'émission si la variable est resynchronisée ou non.

Réception de variable:

Cette demande est mappée sur une primitive de service de niveau de liaison de données

L_RECEIVED.indication(IDENTIFIER)

Le paramètre IDENTIFIER correspond à la valeur de l'attribut *Identifieur* de l'objet *Consumed Variable Local Image*.

Ordre de synchronisation:

Cette demande est mappée sur des primitives de service de niveau de la couche de liaison

L_RECEIVED.indication(IDENTIFIER) ou

L_SENT.indication(IDENTIFIER)

Le paramètre IDENTIFIER correspond à la valeur de l'attribut *Identifieur* de l'objet *Consumed Variable Local Image* (de synchronisation) ou de l'objet *Produced Variable Local Image* (de synchronisation) associé à la variable de synchronisation impliquée dans l'élaboration du statut de validité d'émission.

10.1.3.4 Mapping des demandes et des actions des mécanismes d'élaboration de la liste de récupération

Le contenu de la liste de récupération est géré par un élément de service qui prend en compte tout d'abord, l'ordre de synchronisation associé à la liste, et ensuite, la réception des différentes variables de la liste.

Recovery List Content Management

Initialisation du Contenu de la Liste:

Ces actions sont mappées sur une primitive de service de niveau de la Liaison de Données

L_SPEC_UPDATE.request(SPECIFIED IDENTIFIER, SEQUENCE OF REQUESTED IDENTIFIERS)

Le paramètre SPECIFIED IDENTIFIER correspond à l'attribut *Recovery List Identifieur* de l'objet *List of Variables Local Image*.

Le paramètre SEQUENCE OF REQUESTED IDENTIFIERS correspond à l'attribut *Recovery List Content* de l'objet *List of Variables Local Image*.

Ordre de Synchronisation de la Liste:

Cette action est mappée sur la primitive de service de niveau de liaison de données

L_RECEIVED.indication(IDENTIFIER)

Le paramètre IDENTIFIER correspond à l'attribut *Reference (synchronization) Variable* de l'objet *List of variables Local Image*.

Lorsque les deux objets *Consumed Variable Local Image* et *Produced Variable Local Image* résident dans la même entité d'application, l'ordre de synchronisation est mappé sur la réception de la variable consommée.

Réception de la Liste de Variables:

Cette action est mappée sur la primitive de service de la couche de liaison de données
L_RECEIVED.indication (IDENTIFIER)