

# TECHNICAL REPORT

**ISO/IEC**  
**TR 10024**

First edition  
1992-07-01

---

---

## **Information technology — Telecommunications and information exchange between systems — Formal description of ISO 8073 (Classes 0, 1, 2, 3) in LOTOS**

*Technologies de l'information -- Télécommunications et échange  
d'informations entre systèmes -- Description formelle de l'ISO 8073  
(classes 0, 1, 2 et 3) en LOTOS*



Reference number  
ISO/IEC TR 10024:1992(E)

**AVA** Documents  
Document Sharing Hub

## ISO/IEC TR 10024 : 1992(E)

## Contents

|                                                  |    |
|--------------------------------------------------|----|
| Foreword .....                                   | iv |
| Introduction .....                               | v  |
| 1 Scope .....                                    | 1  |
| 2 Normative references .....                     | 1  |
| 3 Definitions .....                              | 1  |
| 4 Symbols and abbreviations .....                | 1  |
| 5 Conventions .....                              | 1  |
| 6 Requirements .....                             | 2  |
| 7 Introduction of the formal description .....   | 2  |
| 8 Specification parameters data types .....      | 3  |
| 9 Static conformance .....                       | 10 |
| 10 Dynamic conformance .....                     | 11 |
| 11 Service constraints .....                     | 11 |
| 12 Service primitives .....                      | 12 |
| 13 Protocol constraints .....                    | 13 |
| 14 Event Structures .....                        | 15 |
| 15 Identification of transport connections ..... | 18 |
| 16 Provision of a transport connection .....     | 22 |
| 17 Usage of a network connection .....           | 26 |
| 18 Relationship between TPDUs and NSPs .....     | 28 |
| 19 Abstract TPDUs .....                          | 31 |
| 20 Provision and negotiation of options .....    | 41 |
| 21 Relationship between TSPs and TPDUs .....     | 47 |
| 22 Release of a transport connection .....       | 51 |
| 23 Numbering of TPDUs .....                      | 54 |
| 24 Explicit flow control .....                   | 56 |
| 25 Assignment to network connections .....       | 60 |
| 26 Reassignment after failure .....              | 67 |
| 27 Error recovery .....                          | 68 |

© ISO/IEC 1992

All rights reserved. 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 the publisher.

ISO/IEC Copyright Office • Case Postale 56 • CH-1211 Genève 20 • Switzerland

Printed in Switzerland

|                                                                    |                                                       |     |
|--------------------------------------------------------------------|-------------------------------------------------------|-----|
| 28                                                                 | Retention and retransmission .....                    | 70  |
| 29                                                                 | acknowledgement of TPDUs .....                        | 77  |
| 30                                                                 | Resynchronization .....                               | 78  |
| 31                                                                 | Association of TPDUs with transport connections ..... | 81  |
| 32                                                                 | Multiplexing of a network connection .....            | 87  |
| 33                                                                 | Resequencing in class 4 .....                         | 90  |
| 34                                                                 | Use of checksum .....                                 | 90  |
| 35                                                                 | Retransmission in Class 4 .....                       | 90  |
| 36                                                                 | Optional retransmission procedures in class 4 .....   | 90  |
| 37                                                                 | Concatenation of TPDUs .....                          | 90  |
| 38                                                                 | Use of the network service expedited option .....     | 92  |
| 39                                                                 | Treatment of protocol errors .....                    | 93  |
| 40                                                                 | Encoding of TPDUs .....                               | 95  |
| 41                                                                 | Auxiliary definitions .....                           | 118 |
| <b>Annex A</b> Definitions relating to the Transport Service ..... |                                                       | 131 |
| <b>Annex B</b> Definitions relating to the Network Service .....   |                                                       | 139 |
| <b>Annex C</b> Interpretation of ISO 8073 .....                    |                                                       | 144 |
| <b>Annex D</b> Bibliography .....                                  |                                                       | 145 |
| <b>Annex E</b> Index of process definitions .....                  |                                                       | 147 |
| <b>Annex F</b> Index of type definitions .....                     |                                                       | 150 |

## ISO/IEC TR 10024 : 1992(E)

**Foreword**

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) together form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

The main task of technical committees is to prepare International Standards, but in exceptional circumstances a technical committee may propose the publication of a Technical Report of one of the following types:

- type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;
- type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;
- type 3, when a technical committee has collected data of a different kind from that which is normally published as an International Standard ('state of the art', for example).

Technical reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

ISO/IEC TR 10024, which is a technical report of type 2, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

Annexes A, B and C form an integral part of this Technical Report. Annexes D, E and F are for information only.

## Introduction

In view of the complexity and widespread use of Open Systems Interconnection standards it is imperative to have precise and unambiguous definitions of these standards. Formal Description Techniques form an effective approach to providing such definitions. The use of Formal Description Techniques in this area, however, is relatively new, and their application on a wide scale requires an appropriate habituation period. Formal descriptions should be introduced gradually in standards if initially the number of National Bodies that are able to contribute to their development is too small, thus allowing time to gain experience and to develop education material.

Within ISO/IEC JTC 1/SC 6 an ad-hoc group for the formal description in LOTOS of the Transport Standards, viz. the Transport Service ISO 8072 and the Transport Protocol ISO 8073, was established in October 1985. The Formal Description Technique LOTOS, defined in ISO 8807, was still under development at that time within ISO TC 97/SC 21. Following the position of a majority of National Bodies, SC6 decided in January 1988 to progress both formal descriptions as Type 2 Technical Reports. The main reason for not incorporating them into the standards was that National Bodies expressed their current lack of expertise on the subject of Formal Description Techniques. It seemed therefore appropriate to let a period of time pass, during which the formal descriptions can be read and compared with the standards, and after which the status of the formal descriptions can be re-evaluated.

The purpose of this Technical Report is to provide a complete, consistent and unambiguous description of classes 0,1,2 and 3 of ISO 8073. It forms therefore a companion document to ISO 8073. This Technical Report has successfully passed correctness and completeness tests with respect to syntax and static semantics, according to the requirements of clause 3 of ISO 8807. Tools are available that provide interactive simulation facilities, so that confidence can be gained about the correctness of the description with respect to the dynamic semantics (e.g. deadlock-freedom), and about its correctness and completeness with respect to representation of intended behaviour requirements. These tools are still at the stage of prototypes, however.

This Technical Report contains the description of classes 0,1,2,3. Class 4 and NCMS are not yet included. This is due to the limited amount of resources allocated in JTC 1/SC 6 to the work on formal descriptions.

Significant contributions for the application of LOTOS in this work have been provided by Universities and research laboratories.

# Information technology – Telecommunications and information exchange between systems – Formal description of ISO 8073 (Classes 0,1,2,3) in LOTOS

## 1 Scope

This Technical Report describes the classes 0,1,2 and 3 of the OSI Transport protocol, defined in ISO 8073 using the formal description technique LOTOS, which is defined in ISO 8807. Transport protocol class 4 and NCMS are not covered.

## 2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this Technical Report. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this Technical Report are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of the currently valid International Standards.

ISO 7498:1984, *Information Processing Systems – Open Systems Interconnection – Basic Reference Model*.

ISO 8072:1986, *Information Processing Systems – Open Systems Interconnection – Transport service definition*.

ISO/IEC 8073<sup>1)</sup>, *Information Technology – Telecommunications and information exchange between systems – Connection oriented transport protocol specification*.

ISO TR 8509:1987, *Information Processing Systems – Open Systems Interconnection – Service conventions*

ISO 8807:1989, *Information Processing Systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour*.

ISO/IEC TR 10023:1992, *Information Technology – Telecommunications and information exchange between systems – Formal description of ISO 8072 in LOTOS*.

## 3 Definitions

The definitions contained or referred to in clause 3 of ISO 8073 are adopted in this Technical Report.

## 4 Symbols and abbreviations

This Technical Report uses the symbols defined in clause 6 (formal syntax) and Annex A (data type library) of ISO 8807.

The symbols and abbreviations contained in clause 4 of ISO 8073 are employed in this Technical Report. When used without reference, 'the protocol' and 'the service' refer to ISO 8073 resp. ISO 8072.

Usage of other symbols and abbreviations is explained upon their first occurrence.

## 5 Conventions

Informal explanations precede the formal definitions to which they refer, separated from them by a line identical to the first line below. Separation of formal definitions from subsequent informal explanations is indicated by a line identical to the second line below.

---

(\*)

NOTE – This convention is consistent with the rules on delimitation of comments defined for LOTOS in ISO 8807. The formal text is set in Courier and the LOTOS keywords and operators are in boldface. Identifiers from the formal text are set in italics when used in the non-formal text.

The conventions defined in ISO TR 8509 are adopted, but together with the following one: the term "Request" refers both to request and to response service primitives, and the term "Indication" refers both to indication and to confirm service primitives.

---

1) Under revision

## ISO/IEC TR 10024 : 1992(E)

The conventions defined in ISO TR 8509 are adopted, but together with the following one: the term 'Request' refers both to request and to response service primitives, and the term 'Indication' refers both to indication and to confirm service primitives.

Parentheses are omitted in this specification in the following cases, where their removal is allowed by ISO 8807:

- a) In expressions where, due to the fact that binary infix operators associate to the left, no ambiguity arises from removal of parentheses (subclause 6.2.8 of ISO 8807). Thus, for example, the expression '*x gt 0* and (*y lt 5*)' is specified as '*x gt 0* and (*y lt 5*)'.
- b) In expressions where, due to the priority of LOTOS behavioural operators, no ambiguity arises from removal of parentheses. Thus, for example, '*g; (stop)*' is specified as '*g; stop*'.

The LOTOS standard allows the use of explicit sort specification for simple-expression using the 'ofsort' terms. Such 'ofsort' terms are present in the specification only in those cases where their absence would give rise to ambiguity.

## 6 Requirements

This Technical Report complies with the requirements defined by clause 3 of ISO 8807.

No conformity requirements are defined by this Technical Report, but the conformance requirements defined by ISO 8073 are represented formally in LOTOS.

## 7 Introduction of the formal description

The present specification is designed on basis of two major requirements, which together provide the appropriate framework for the formal representation of the transport protocol architecture by way of a formal specification of a generic transport protocol entity:

- a) the formal specification is to be provably consistent with the formal description in LOTOS of the OSI connection-oriented transport service ISO 8072 (TS FD for short, that is available in ISO TR 10023), assuming a correct formal description in LOTOS of the OSI connection-oriented network service ISO 8348 (NS FD for short),
- b) the specification applies to any transport protocol entity which is claimed to comply with the requirements defined in the conformance clause of the protocol.

Figure 1 shows the resulting structure of the specification.

A formal data type is provided for the implementor's declaration of classes and options that are defined in the conformance clause of the protocol. Such a declaration is formally represented by a parameter of the specification. A boolean-valued function can be applied to the value of this parameter, that determines whether or not the value satisfies the static conformance requirements of the protocol (see clause 9).

Only if the static conformance requirements are satisfied, does the specification describe active behaviour, viz. it provides the implementor or tester with an abstract model of the protocol dynamic conformance requirements (see clause 10)

|                                 |                                                                                                                                                                |
|---------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>specification</b>            | parameters: gates: t,n<br>SAP addresses<br>Implementation options                                                                                              |
| <b>global type definitions:</b> | library data types<br>SAP address types<br>Implementation option types<br>Auxiliary data types                                                                 |
| <b>behaviour definition:</b>    | if    static conformance requirements are<br>satisfied by the implementation options<br><br>then   dynamic conformance requirements<br><br>else   no behaviour |
| <b>where local definitions:</b> | static conformance specification<br>dynamic conformance specification                                                                                          |
| <b>endspec</b>                  |                                                                                                                                                                |

Figure 1 - Structure of the specification

The  $t$  (resp.  $n$ ) gate represents the transport (resp. network) service boundary accessed by the entity. It models the totality of TSAPs (resp. NSAPs) at which the entity interacts with the TS user (resp. NS provider). Each TSAP (resp. NSAP) is uniquely identified by a TSAP address (resp. NSAP address) out of the set  $tas$  (resp.  $nas$ ).

Proper co-operation between session and transport entities (and transport and network entities, resp.), within one open system, is ensured by assigning the same  $tas$  to the session and transport entities (and the same  $nas$  to the transport and network entities, resp.) residing in the same open system. This implies that a transport entity cannot co-operate with a session entity if they reside in different open systems.

The event structure at gate  $t$  consists of a triple of values, resp. of sorts  $TAddress$ ,  $TCEI$ ,  $TSP$  (see clause 8). The first value identifies the TSAP where the interaction occurs. The second value identifies the TCEP, within that TSAP, where the interaction occurs. The third value is the Transport Service Primitive (TSP) executed in the interaction. A similar structure is shown by events at gate  $n$ .

These event structures are the same as in the respective service formal descriptions, with the exception that this Technical Report describes a more refined, non-atomic execution of T-DATA primitives. This is required by correctness and generality of representation of the requirements relating to segmenting and flow control procedures. Clearly, also the data type definition of TSPs needed corresponding refinements: the definition presented in this Technical Report is a conservative extension of the definition found in the formal description in LOTOS of the transport service.

Full account is taken of the multiplicity aspects of the protocol. The behaviour of a never terminating transport entity is described.

According to the definition of process  $TPEntity$  (see clause 10), some of the component processes describe constraints that apply to, and depend upon, the behaviour of the protocol entity at only one of the two service boundaries. This class of constraints will be referred to as 'service constraints' (see clause 11), whilst the term 'protocol constraints' will refer to those which are described by the other components (see clause 12).

The service constraints ensure, for instance, that the address component of an interaction at  $t$  (resp.  $n$ ) is a member of the set  $tas$  (resp.  $nas$ ), that the identification of a connection by means of a connection endpoint identifier is unique within the scope of any given address, that the entity is ready to accept and support at least one TC and at least one NC, and so on.

The further decomposition of the protocol constraints exploits the usage of internal gates. Notice that, according to the formal semantics of LOTOS, the presence of internal gates in the specification by no way constrains the internal structure of any implementation, inasmuch as the specification requirements apply to the behaviour observable at service access points only.

The introduction of these internal gates and the definition of the related event structures (see clause 14) enable a clear separation of concerns between:

- a) independent constraints on the provision of TCs (see clause 15, clause 16, clause 18, clause 20 through clause 24, clause 33 through clause 36, clause 39, clause 40),
- b) independent constraints on the usage of NCs (see clause 17, clause 32, clause 37, clause 38),
- c) dependencies between provision of TCs and usage of NCs, such as those relating to assignment of TCs to NCs (see clause 25), error recovery procedures (see clause 26 through clause 30), and association of TPDU to TCs (see clause 31).

A distinction is made between the definition of abstract TPDU structures (see clause 19), where abstraction is made from the TPDU information that relates to TC identification, encoding, and encoded TPDU, which are octet strings where all of the TPDU information is fully specified (see clause 40).

Some auxiliary definitions are finally presented (see clause 41), which do not directly represent constraints found in the protocol, but are frequently employed in the preceding formal definitions. Processes that have a name beginning with *Run* are all defined in clause 41.

Except for the definitions of the specification parameter types, definitions shared with the TS FD, ISO TR 10023, and sharable with a NS FD are presented in Annex A and Annex B respectively. The data types referred to in the **library** construct are imported from the LOTOS library of data types.

---

```

specification TransportProtocolEntity[t, n]
  (tas : TAddresses, nas : NAddresses, tpeo : TPEOptions) : noexit
library Set, String, NaturalNumber, NatRepresentations, DecNatRepr, OctetString, Octet, Bit,
  BitNatRepr, Boolean, FBoolean, Element
endlib
  (*

```

---

## 8 Specification parameters data types

### 8.1 General

Two of the three specification parameters (see clause 7) have simple structures: finite sets of addresses and are presented in 8.2.

## ISO/IEC TR 10024 : 1992(E)

The options parameter contains the description of the optional procedures and their parameters, viz. supported classes, supported roles (*initiator* or *responder*), maximum TPDUsizes for each supported class and the optional procedures listed in table 9 of the protocol.

It has a slightly more complicated structure, which can be summarized as follows: a value of sort *TPEOptions* is a 4-tuple (see 8.3), where the components correspond to the requirements mentioned in subclause 14.6 of the protocol. The component structures are respectively described in 8.3.1 through 8.3.4.

## 8.2 Address sets

Although some these definitions below are shared with the TS FD or can be shared with the NS FD, these definitions are presented here, instead of in clause A.2, resp clause B.2, to allow description of the specification parameters.

---

```

type TransportAddress
is GeneralIdentifier renamedby
sortnames
  TAddress for Identifier
opnnames
  SomeTAddress for SomeIdentifier
  AnotherTAddress for AnotherIdentifier
endtype (* TransportAddress *)

type TransportAddresses
is Set actualizedby TransportAddress using
sortnames
  TAddress for Element
  Bool for FBool
  TAddresses for Set
endtype (* TransportAddresses *)

type NetworkAddress
is GeneralIdentifier renamedby
sortnames
  NAddress for Identifier
opnnames
  SomeNAddress for SomeIdentifier
  AnotherNAddress for AnotherIdentifier
endtype (* NetworkAddress *)

type NetworkAddresses
is Set actualizedby NetworkAddress using
sortnames
  NAddress for Element
  Bool for FBool
  NAddresses for Set
endtype (* NetworkAddresses *)

```

---

## 8.3 Options

A value of sort *TPEOptions* is a 4-tuple constructed by the function *TPEOptions*. The functions *Classes*, *Roles*, *OptionalProcedures* and *MaxTPDUSizeMap* are projection functions, viz. application of each of them to a quadruple yields the value of one distinct component. The definition of boolean equality is straightforward.

---

```

type TPEOptions
is Classes, TPEntityRoles, OptionalProcedures, MaxTPDUSizeMap, PrMaxTPDUSize
sorts
  TPEOptions
opns
  TPEOptions          : Classes, TPERoles, OptionalProcedures,
  MaxTPDUSizeMap, PrMaxTPDUSize      -> TPEOptions
  Classes             : TPEOptions          -> Classes
  Roles               : TPEOptions          -> TPERoles
  OptionalProcedures  : TPEOptions          -> OptionalProcedures
  MaxTPDUSizeMap      : TPEOptions          -> MaxTPDUSizeMap
  PrMaxTPDUSize       : TPEOptions          -> PrMaxTPDUSize
  _eq_, _ne_         : TPEOptions, TPEOptions -> Bool
eqns
forall

```

---

```

t, t1, t2 : TPEOptions, c, c1 : Classes, r, r1 : TPERoles, p, p1 : OptionalProcedures,
s, s1 : MaxTPDUSizeMap, pm, pm1 : PrMaxTPDUSize
ofsort Classes
  Classes(TPEOptions(c, r, p, s, pm)) = c;
ofsort TPERoles
  Roles(TPEOptions(c, r, p, s, pm)) = r;
ofsort OptionalProcedures
  OptionalProcedures(TPEOptions(c, r, p, s, pm)) = p;
ofsort MaxTPDUSizeMap
  MaxTPDUSizeMap(TPEOptions(c, r, p, s, pm)) = s;
ofsort PrMaxTPDUSize
  PrMaxTPDUSize(TPEOptions(c, r, p, s, pm)) = pm;
ofsort Bool
  TPEOptions(c, r, p, s, pm) eq TPEOptions(c1, r1, p1, s1, pm1)
    = c eq c1 and (r eq r1) and (p eq p1) and (s eq s1) and (pm eq pm1);
  t1 ne t2 = not(t1 eq t2);
endtype (* TPEOptions *)
(*)

```

### 8.3.1 Protocol classes

The five protocol classes are represented by corresponding constants of sort *Class*. These are ordered by means of the auxiliary function *h*, that maps classes to natural numbers. *Classes* defines finite sets of protocol classes, enriched with the function *HigherClasses* which yields the set of classes higher than a given class. This function is made use of in the description of class negotiation (see 20.2.2).

```

type Class
is NaturalNumber
sorts
  Class
opns
  Class0, Class1, Class2, Class3, Class4 :          -> Class
  _eq_, _ne_, _le_, _lt_, _ge_, _gt_ : Class, Class -> Bool
  h : Class                                          -> Nat
eqns
forall
  x, y : Class
ofsort Nat
  h(Class0) = 0;
  h(Class1) = succ(h(Class0));
  h(Class2) = succ(h(Class1));
  h(Class3) = succ(h(Class2));
  h(Class4) = succ(h(Class3));
ofsort Bool
  x eq y = h(x) eq h(y);
  x ne y = h(x) ne h(y);
  x le y = h(x) le h(y);
  x lt y = h(x) lt h(y);
  x ge y = h(x) ge h(y);
  x gt y = h(x) gt h(y);
endtype (* Class *)
type BasicClasses
is Set actualizedby Class using
sortnames
  Class for Element
  Bool for FBool
  Classes for Set
endtype (* BasicClasses *)
type Classes
is BasicClasses
opns
  HigherClasses : Class -> Classes
eqns
forall
  c, d : Class
ofsort Classes

```

## ISO/IEC TR 10024 : 1992(E)

```

HigherClasses(Class4) = {};
h(d) = succ(h(c)) => HigherClasses(c) = Insert(d, HigherClasses(d));
endtype (* Classes *)

```

---

## 8.3.2 Entity roles

An entity can either initiate CR TPDUs (*Initiator*), respond to CR TPDUs (*Responder*) or both (see 8.4 for *TwoTuplet*).

```

type TPEntityRole
is TwoTuplet renamedby
sortnames
  TPERole for Tuplet
opnnames
  Initiator for TheOne
  Responder for TheOther
endtype (* TPEntityRole *)

type TPEntityRoles
is Set actualizedby TPEntityRole using
sortnames
  TPERole for Element
  Bool for FBool
  TPERoles for Set
endtype (* TPEntityRoles *)

```

---

## 8.3.3 Optional procedures

*OptionalProcedure* defines constants for those procedures that have some 'optional' entry in table 9 of the protocol. The auxiliary function *h* enables a concise specification of boolean equality. The boolean functions *AppliesIn* specify whether a given optional procedure applies in a given class, or in some class of a given set of classes.

NOTE – The two functions *AppliesIn* differ in the second argument.

```

type OptionalProcedure
is Classes, DecNatRepr
sorts
  OptionalProcedure
opns
  TPDUWithoutChecksum, NoFlowControl, ExtendedFormats, UseOfNSReceiptConfirmation,
  UseOfNSExpedited, UseOfSack, UseOfRack :
    h : OptionalProcedure -> OptionalProcedure
    h : OptionalProcedure -> Nat
    _eq_, _ne_ : OptionalProcedure, OptionalProcedure -> Bool
    _AppliesIn_ : OptionalProcedure, Class -> Bool
    _AppliesIn_ : OptionalProcedure, Classes -> Bool
eqns
forall
  p, pl : OptionalProcedure, c : Class, cs : Classes
ofsort Nat
  h(TPDUWithoutChecksum) = 0;
  h(NoFlowControl) = succ(h(TPDUWithoutChecksum));
  h(ExtendedFormats) = succ(h(NoFlowControl));
  h(UseOfNSReceiptConfirmation) = succ(h(UseOfNSReceiptConfirmation));
  h(UseOfNSExpedited) = succ(h(UseOfNSReceiptConfirmation));
  h(UseOfSack) = succ(h(UseOfNSExpedited));
  h(UseOfRack) = succ(h(UseOfSack));
ofsort Bool
  p eq pl = h(p) eq h(pl);
  p ne pl = not(p eq pl);
  TPDUWithoutChecksum AppliesIn c = c eq Class4;
  NoFlowControl AppliesIn c = c eq Class2;
  ExtendedFormats AppliesIn c = c ge Class2;
  UseOfNSReceiptConfirmation AppliesIn c = c eq Class1;
  UseOfNSExpedited AppliesIn c = c eq Class1;
  UseOfSack AppliesIn c = c eq Class4;
  UseOfRack AppliesIn c = c eq Class1 or (c ge Class3);

```

---

```

p AppliesIn {} = false;
c IsIn cs, p AppliesIn c => p AppliesIn cs = true;
c NotIn cs, not(p AppliesIn c) => p AppliesIn Insert(c, cs) = p AppliesIn cs;
endtype (* OptionalProcedure *)

type OptionalProcedures
is Set actualizedby OptionalProcedure using
sortnames
  OptionalProcedure for Element
  Bool for FBool
  OptionalProcedures for Set
endtype (* OptionalProcedures *)
(*)

```

#### 8.3.4 Maximum TPDU sizes

*MaxTPDUSize* defines seven constants that represent the values listed in subclause 14.6 of the protocol, and the functions *TPDUSize*, which yields the corresponding natural number, and *MaxTPDUSizeFor*, which yields the maximum TPDU size defined by the protocol for each class (see subclause 13.3.4.b of the protocol). Equality and order of the constants are respectively specified by means of equality and order of their corresponding *TPDUSize* value.

*MaxTPDUSizes* defines sets of *MaxTPDUSize* values, and the boolean functions *MaxIn*, which tells whether a given *MaxTPDUSize* value is the maximum in a given set of such values, and *Maximal*, which tells whether a given set of *MaxTPDUSize* values contains all the values lower than or equal to the maximum value that it contains.

*MaxTPDUSizeInClass* defines pairs  $\langle \text{MaxTPDUSize} \times \text{Class} \rangle$  (see 8.4 for the definition of the generic type *Pair*). *BasicMaxTPDUSizeMap* defines sets of these pairs as values of sort *MaxTPDUSizeMap*.

In fact, a value of sort *MaxTPDUSizeMap* can be viewed as a binary relation between *MaxTPDUSize* and *Class*: in this view, the type *MaxTPDUSizeMap* endows the basic type with the functions *Classes*, which yields the range of the argument relation, and *MaxTPDUSizesOf* which, for given class and relation, yields the subset of the domain of the given relation characterized by the given class.

NOTE – The expression  $1 + (2 + \text{Dec}(8))$  below denotes a string of decimal digits, where '+' denotes digit concatenation with a digit string. The corresponding natural number value (128, in this case) is obtained by application of the function *NatNum*.

```

type MaxTPDUSize
is DecNatRepr, Class
sorts
  MaxTPDUSize
opns
  Max1, Max2, Max3, Max4, Max5, Max6, Max7 :                -> MaxTPDUSize
  MaxTPDUSizeFor : Class                                     -> MaxTPDUSize
  TPDUSize : MaxTPDUSize                                     -> Nat
  _eq_, _ne_, _le_, _lt_, _ge_, _gt_ : MaxTPDUSize, MaxTPDUSize -> Bool
eqns
forall
  c : Class, m, m1 : MaxTPDUSize
ofsort MaxTPDUSize
  MaxTPDUSizeFor(Class0) = Max5;
  c ne Class0 => MaxTPDUSizeFor(c) = Max7;
ofsort Nat
  TPDUSize(Max1) = NatNum(1 + (2 + Dec(8)));
  TPDUSize(Max2) = TPDUSize(Max1) + TPDUSize(Max1);
  TPDUSize(Max3) = TPDUSize(Max2) + TPDUSize(Max2);
  TPDUSize(Max4) = TPDUSize(Max3) + TPDUSize(Max3);
  TPDUSize(Max5) = TPDUSize(Max4) + TPDUSize(Max4);
  TPDUSize(Max6) = TPDUSize(Max5) + TPDUSize(Max5);
  TPDUSize(Max7) = TPDUSize(Max6) + TPDUSize(Max6);
ofsort Bool
  m eq m1 = TPDUSize(m) eq TPDUSize(m1);
  m ne m1 = TPDUSize(m) ne TPDUSize(m1);
  m le m1 = TPDUSize(m) le TPDUSize(m1);
  m lt m1 = TPDUSize(m) lt TPDUSize(m1);
  m ge m1 = TPDUSize(m) ge TPDUSize(m1);
  m gt m1 = TPDUSize(m) gt TPDUSize(m1);
endtype (* MaxTPDUSize *)

type BasicMaxTPDUSizes
is Set actualizedby MaxTPDUSize using

```

\*)

## ISO/IEC TR 10024 : 1992(E)

```

sortnames
  MaxTPDUSize for Element
  Bool for FBool
  MaxTPDUSizes for Set
endtype (* BasicMaxTPDUSizes *)

type MaxTPDUSizes
is BasicMaxTPDUSizes

opns
  _MaxIn_           : MaxTPDUSize, MaxTPDUSizes -> Bool
  Maximal           : MaxTPDUSizes             -> Bool

eqns
forall
  m, m1 : MaxTPDUSize, ms : MaxTPDUSizes
ofsort Bool
  m NotIn ms => m MaxIn ms = false;
  m MaxIn Insert(m, {}) = true;
  m NotIn ms, m1 MaxIn ms => m MaxIn Insert(m, ms) = m gt m1;
  Maximal({}) = true;
  Maximal(Insert(m, {})) = m eq Max1;
  m NotIn ms, m1 MaxIn ms, Maximal(ms)
    => Maximal(Insert(m, ms)) = TPDUSize(m) eq (TPDUSize(m1) + TPDUSize(m1));
endtype (* MaxTPDUSizes *)

type MaxTPDUSizeClassPair
is Pair actualizedby MaxTPDUSize, Class using
sortnames
  MaxTPDUSize for Element
  Class for Element2
  Bool for FBool
  MaxTPDUSizeInClass for Pair
opnnames
  MaxTPDUSize for First
  Class for Second
  MaxTPDUSizeInClass for Pair
endtype (* MaxTPDUSizeClassPair *)

type BasicMaxTPDUSizeMap
is Set actualizedby MaxTPDUSizeClassPair using
sortnames
  MaxTPDUSizeInClass for Element
  Bool for FBool
  MaxTPDUSizeMap for Set
endtype (* BasicMaxTPDUSizeMap *)

type MaxTPDUSizeMap
is BasicMaxTPDUSizeMap, MaxTPDUSizes, Classes

opns
  Classes           : MaxTPDUSizeMap           -> Classes
  _MaxTPDUSizesOf_  : Class, MaxTPDUSizeMap     -> MaxTPDUSizes

eqns
forall
  m : MaxTPDUSize, ms : MaxTPDUSizeMap, c, c1 : Class
ofsort Classes
  Classes({}) = {};
  Classes(Insert(MaxTPDUSizeInClass(m, c), ms)) = Insert(c, Classes(ms));
ofsort MaxTPDUSizes
  c NotIn Classes(ms) => c MaxTPDUSizesOf ms = {};
  c MaxTPDUSizesOf Insert(MaxTPDUSizeInClass(m, c), ms) = Insert(m, c MaxTPDUSizesOf ms);
  c ne c1 => c MaxTPDUSizesOf Insert(MaxTPDUSizeInClass(m, c1), ms) = c MaxTPDUSizesOf ms;
endtype (* MaxTPDUSizeMap *)
(*

```

The preferred maximum tpdusize parameter is expressed as a multiple of 128 octets.

```

type PrMaxTPDUSize
is BasicPrMaxTPDUSize, NatRepresentations
opns
  NatNum           : PrMaxTPDUSize           -> Nat
  _le_             : PrMaxTPDUSize, prMaxTPDUSize -> Bool

```

```

eqns
forall
  pr, pr1 : PrMaxTPDUSize
ofsort Nat
  NatNum(NoPreference) = 0;
  NatNum(Next128(pr)) = NatNum(pr) + NatNum(1 + (2 + Dec(8)));
ofsort Bool
  pr le pr1 = NatNum(pr) le NatNum(pr1);
endtype (* PrMaxTPDUSize *)
type BasicPrMaxTPDUSize
is GeneralIdentifier renamedby
sortnames
  PrMaxTPDUSize for Identifier
opnnames
  NoPreference for SomeIdentifier
  Next128 for AnotherIdentifier
endtype (* BasicPrMaxTPDUSize *)
(*

```

#### 8.4 Auxiliary definitions

This clause contains the definitions of *TwoTuplet*, *Pair* and *GeneralIdentifier* needed in constructing the definitions from clauses 8.2 and 8.3.

*Element2* is needed in the construction of *Pair* and is a renaming of the library type *Element*.

*Pair* is a tuple  $\langle \textit{Element} \times \textit{Element2} \rangle$  with extended equality and operations for extracting its components.

*TwoTuplet* specifies a data type whose sort (*Tuplet*) can have 2 distinct values (*TheOne* and *TheOther*) together with equality.

*GeneralIdentifier* specifies an infinite number of identifiers together with equality.

NOTE – The auxiliary function *h* is introduced to simplify the specification of operation specifying equality. This technique will be used throughout this Technical Report.

```

type Element2
is Element renamedby
sortnames
  Element2 for Element
endtype (* Element2 *)
type Pair
is Boolean, Element, Element2
sorts
  Pair
opns
  Pair          : Element, Element2      -> Pair
  First         : Pair                   -> Element
  Second        : Pair                   -> Element2
  _eq_, _ne_    : Pair, Pair             -> Bool
eqns
forall
  e1 : Element, e2 : Element2, p, p1 : Pair
ofsort Element
  First(Pair(e1, e2)) = e1;
ofsort Element2
  Second(Pair(e1, e2)) = e2;
ofsort Bool
  p = p1 => p eq p1 = true;
  First(p) ne First(p1) => p eq p1 = false;
  Second(p) ne Second(p1) => p eq p1 = false;
  p ne p1 = not(p eq p1);
endtype (* Pair *)
type TwoTuplet
is Boolean, NaturalNumber
sorts
  Tuplet
opns
  TheOne, TheOther      :                -> Tuplet
  _eq_, _ne_           : Tuplet, Tuplet   -> Bool

```

## ISO/IEC TR 10024 : 1992(E)

```

h                               : Tuplet                               -> Nat
eqns
forall
  u, v : Tuplet
ofsort Nat
  h(TheOne) = 0;
  h(TheOther) = Succ(h(TheOne));
ofsort Bool
  u eq v = h(u) eq h(v);
  u ne v = not(u eq v);
endtype (* TwoTuplet *)
type GeneralIdentifier
is Boolean
sorts
  Identifier
opns
  SomeIdentifier                :                               -> Identifier
  AnotherIdentifier             : Identifier                   -> Identifier
  _eq_, _ne_                    : Identifier, Identifier       -> Bool
eqns
forall
  a, al : Identifier
ofsort Bool
  SomeIdentifier eq SomeIdentifier = true;
  AnotherIdentifier(a) eq SomeIdentifier = false;
  SomeIdentifier eq AnotherIdentifier(a) = false;
  AnotherIdentifier(a) eq AnotherIdentifier(al) = a eq al;
  a ne al = not(a eq al);
endtype (* GeneralIdentifier *)
(*

```

## 9 Static conformance

The static validity of the implementor's conformance claim is represented by the truth value of the function *Conforms* applied to the specification parameter (see 8.3 for *TPEOptions* and related functions). The claim is valid only if four requirements are satisfied, respectively applied to:

- set of classes implemented, according to subclauses 14.2, 14.3, 14.4 and 14.6.a of the protocol,
- set of roles (i.e. initiator, responder) implemented, according to subclause 14.6.c of the protocol,
- set of optional procedures for which conformance is claimed, according to subclause 14.6.d and table 9 of the protocol,
- maximum TPDU sizes implemented, according to subclause 14.6.e of the protocol.

NOTE – The boolean function *AppliesIn* (see 8.3.3) only represents - according to table 9 of the protocol - optionality of a given procedure for some of the classes implemented. Moreover, this function is only defined for those procedures that have some 'optional' entry in table 9 of the protocol: the requirements following from the 'mandatory' and 'not applicable' entries pertain to provision and negotiation of options, according to the classes implemented (see clause 20).

### behaviour

[Conform(tpeo)] -> TPEntity [t, n] (tas, nas, tpeo)

### where

type TPStaticConformance

is TPEOptions

### opns

|                           |                               |         |
|---------------------------|-------------------------------|---------|
| Conform                   | : TPEOptions                  | -> Bool |
| ClassesConform            | : Classes                     | -> Bool |
| RolesConform              | : TPERoles                    | -> Bool |
| OptionalProceduresConform | : OptionalProcedures, Classes | -> Bool |
| MaxTPDUSizeMapConform     | : MaxTPDUSizeMap, Classes     | -> Bool |
| PrMaxTPDUSizeConform      | : PrMaxTPDUSize               | -> Bool |

### eqns

### forall

cs : Classes, c : Class, rs : TPERoles, ops : OptionalProcedures,  
ms : MaxTPDUSizeMap, op : OptionalProcedure, m : MaxTPDUSize, pm : PrMaxTPDUSize

ofsort Bool

```

Conform(TPEOptions(cs, rs, ops, ms, pm))
  = ClassesConform(cs) and RolesConform(rs)
    and OptionalProceduresConform(ops, cs) and (Classes(ms) eq cs)
    and MaxTPDUSizeMapConform(ms, cs) and PrMaxTPDUSizeConform(pm);
ClassesConform(cs)
  = Class0 IsIn cs or (Class2 IsIn cs) and (Class3 IsIn cs or (Class4 IsIn cs)
    implies (Class2 IsIn cs)) and (Class1 IsIn cs implies (Class0 IsIn cs));
RolesConform(rs) = rs ne {};
OptionalProceduresConform({} of OptionalProcedures, cs) = true;
OptionalProceduresConform(Insert(op, ops), cs)
  = op AppliesIn cs and OptionalProceduresConform(ops, cs);
MaxTPDUSizeMapConform(ms, {}) = true;
m MaxIn (c MaxTPDUSizesOf ms)
  => MaxTPDUSizeMapConform(ms, Insert(c, cs))
  = m le MaxTPDUSizeFor(c) and Maximal(c MaxTPDUSizesOf ms) and MaxTPDUSizeMapConform(ms, cs);
c IsIn cs, c NotIn Classes(ms) => MaxTPDUSizeMapConform(ms, cs) = false;
PrMaxTPDUSizeConform(pm) = pm ne NoPreference;
endtype (* TPStaticConformance *)

```

(\*

## 10 Dynamic conformance

Process *TPEntity* specifies the valid behaviour of transport protocol entities. This process describes the relationship between provision of TCs and usage of NCs, subject to service constraints both at the TS boundary and at the NS boundary (see Annex B).

The TS constraints are expressed by processes *TCIdentification* and *TPETCAcceptance*: the former is shared with the TS FD (see clause A.3), the latter is very similar to the *TCAcceptance* process of the TS FD (cf. ISO TR 10023) but for the presence of a bound on the set of transport addresses where TCs may be accepted (this bound here represents the fact that the transport entity is confined within an end-system).

The NS constraints are similarly expressed by processes *NCIdentification* and *TPENCAcceptance*: again, only the former can be shared with a NS FD (see clause B.3), whilst the latter, in this case, would differ from a *NCAcceptance* process of a NS FD also w.r.t. specification of non-determinism; this is so because the NS requirements apply to the NS provider, whilst here the NS user is described.

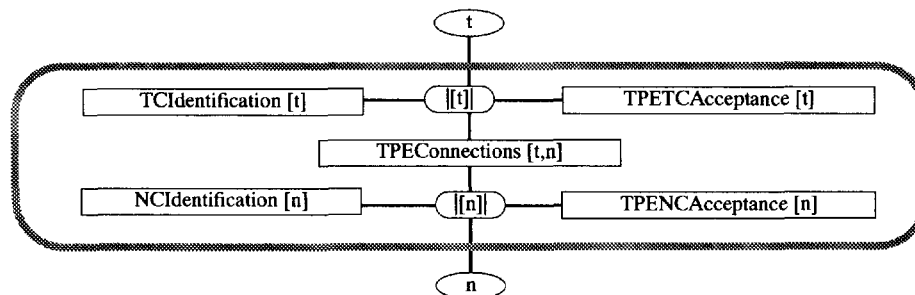


Figure 2 - Constraints on the TP entity behaviour

```

process TPEntity[t, n](tas : TAddresses, nas : NAddresses, tpeo : TPEOptions) : noexit :=
  TPEConnections [t, n] (tpeo)
  ||
  (TCIdentification [t] || TPETCAcceptance [t] (tas))
  ||
  (NCIdentification [n] || TPENCAcceptance [n] (nas))
endproc (* TPEntity *)

```

(\*

## 11 Service constraints

*TCIdentification* and *NCIdentification*: see resp. clause A.3 and clause B.3.

*TPETCAcceptance* specifies the following constraints:

- TS primitives may only be exchanged at transport addresses accessed by the protocol entity, and
- at all times at least one TC endpoint is provided that allows a T-CONNECT request to occur.

## ISO/IEC TR 10024 : 1992(E)

*TPENCAcceptance* specifies the following constraints:

- a) NS primitives may only be exchanged at network addresses accessed by the protocol entity, and
- b) at all times at least one NC endpoint is provided that allows an N-CONNECT indication to occur.

---

```

process TPETCAcceptance[t](tas : TAddresses) : noexit:=
choice tias : TIds
  []
  [tias ne {} and (Addresses(tias) IsSubsetOf tas)]->
  i ;
  t?ta : TAddress ?tcei : TCEI ?tsp : TSP
  [ta IsIn tas and (IsTCONreq(tsp) implies (TId(ta, tcei) IsIn tias))] ;
  TPETCAcceptance [t] (tas)
endproc (* TPETCAcceptance *)
process TPENCAcceptance[n](nas : NAddresses) : noexit:=
choice nias : NIds
  []
  [nias ne {} and (Addresses(nias) IsSubsetOf nas)]->
  i ;
  n?NA : NAddress ?ncei : NCEI ?nsp : NSP
  [NA IsIn nas and (IsNCONind(nsp) implies (NId(NA, ncei) IsIn nias))] ;
  TPENCAcceptance [n] (nas)
endproc (* TPENCAcceptance *)
type TPETIdAcceptance
is TransportAddresses, TCEIdentifications
opns
  Addresses : TIds -> TAddresses
eqns
forall
  ti : TId, tis : TIds
ofsort TAddresses
  Addresses({}) = {};
  Addresses(Insert(ti, tis)) = Insert(TA(ti), Addresses(tis));
endtype (* TPETIdAcceptance *)
type TPENIdAcceptance
is NetworkAddresses, NCEIdentifications
opns
  Addresses : NIds -> NAddresses
eqns
forall
  ni : NId, nis : NIds
ofsort NAddresses
  Addresses({}) = {};
  Addresses(Insert(ni, nis)) = Insert(NA(ni), Addresses(nis));
endtype (* TPENIdAcceptance *)
  (*

```

---

## 12 Service primitives

The *TPEntityTransportServicePrimitive* data type definition is a refinement (conservative extension) of the service definition *TransportServicePrimitive* (see clause A.2).

*TDTOreq* and *TDTOind* represent transfer of an octet of the user data parameter of T-DATA request and indication primitives, respectively. *TDTEreq* and *TDTEind* represent termination of execution of T-DATA primitives.

---

```

type TPEntityTransportServicePrimitive
is TransportServicePrimitive
opns
  TDTOreq, TDTOind : Octet -> TSP
  TDTEreq, TDTEind : -> TSP
  IsTDTO, IsTDTE, IsTDTOreq, IsTDTOind, IsTDTEreq, IsTDTEind : TSP -> Bool
eqns
forall
  u : Octet, d : OctetString, t : TSP, s : TDTASubsort

```

```

ofsort Bool
  IsTDTO(t) = IsTDTOreq(t) or IsTDTOind(t);
  IsTDTE(t) = IsTDTEreq(t) or IsTDTEind(t);
  IsTDTOreq(t)
    = IsTDTreq(t) and (Length(UserData(t)) eq succ(0)) and not(TEOTrequest Terminates t);
  IsTDTOind(t)
    = IsTDTind(t) and (Length(UserData(t)) eq succ(0)) and not(TEOTindication Terminates t);
  IsTDTEreq(t) = IsTDTreq(t) and (UserData(t) eq <>);
  IsTDTEind(t) = IsTDTind(t) and (UserData(t) eq <>);
  TEOTrequest Terminates TDTEreq = true;
  TEOTindication Terminates TDTEind = true;
  TEOTrequest Terminates TDTEind = false;
  TEOTindication Terminates TDTEreq = false;
  s Terminates TDTOreq(u) = false;
  s Terminates TDTOind(u) = false;
ofsort OctetString
  UserData(TDTOreq(u)) = Octet(u);
  UserData(TDTOind(u)) = Octet(u);
  UserData(TDTEreq) = <>;
  UserData(TDTEind) = <>;
ofsort TSPSubsort
  Subsort(TDTOreq(u)) = TDATArequest;
  Subsort(TDTOind(u)) = TDATAindication;
  Subsort(TDTEreq) = TDATArequest;
  Subsort(TDTEind) = TDATAindication;
endtype (* TPEntityTransportServicePrimitive *)
  (*

```

## 13 Protocol constraints

### 13.1 General

The decomposition of the behaviour describing dynamic conformance finds its most complex component in the process which describes the relationship between provision of TCs and usage of NCs.

The form of the definition of *TPEConnections*, and the design of event structures at the internal gates, are aimed at facilitating the representation of distinct protocol constraints by distinct processes:

- at *p* TPDU transfers are described, together with related information (see 14.3.1),
- at *a* (re)assignment to NC occurs (see 14.3.2),
- at *d* deassignment from NC occurs (see 14.3.3),
- at *s* NC reset are indicated (see 14.3.4).

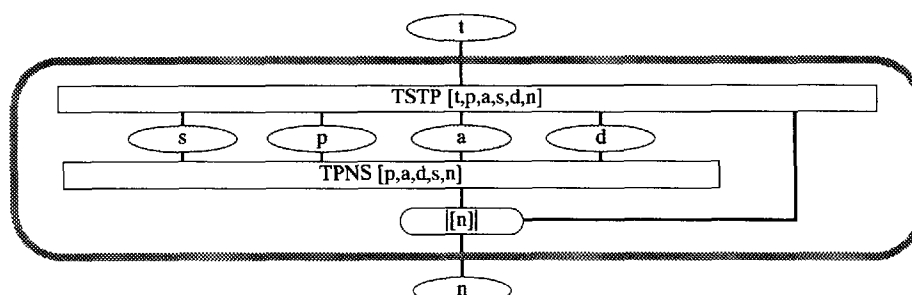


Figure 3 - Structure of process TPEConnections

```

process TPEConnections[t, n](tpeo : TPEOptions) : noexit :=
hide p, a, d, s
  in TSTP [t, p, a, d, s, n] (tpeo)
    | [p, a, d, s, n] |
    TPNS [p, a, d, s, n]
endproc (* TPEConnections *)
  *)

```

## 13.2 Process structures

### 13.2.1 General

With reference to the structure depicted in Figure 3, the following decomposition of *TPEConnections* is pursued:

- a) *TSTP*: constraints that refer to provision of TCs; this includes: TC identification, class and options negotiation, association of TPDU with TCs, segmenting and reassembling, flow control, TC release, splitting and recombining, error recovery procedures, handling of protocol errors, encoding of TPDU,

NOTE - this process accesses the NS boundary for the only reason that some cases of the aforementioned procedures (viz. TPDU acknowledgement by the Receipt Confirmation option,

- b) *TPNS*: constraints that refer to the usage of NCs; this includes: multiplexing, concatenation and separation, usage of network expedited.

### 13.2.2 Provision of transport connections

The definition of *TSTP* (see Figure 4) follows the distinction between constraints that can be expressed locally to each TC (process TCs, consisting of an unbounded number of parallel, unsynchronized instances of process TC - see clause 16), and constraints the formulation of which needs global information, viz. data relating to the whole set of active TCs. Two processes relate to the second class of constraints:

- a) *LRefs* (see clause 15): constraints on unique identification of TCs by local references; also frozen references are represented by this process,
- b) *TPDUAssociation* (see clause 31): constraints, and related TPDU handling, that apply to association of TPDU with TC, in accordance with subclause 6.9.1.4.2 of the protocol.

NOTE - 1 *NoTC* (see clause 31) describes handling of received TPDU that are not associated with any TC.

NOTE - 2 *TCNAccess* (see 25.6) is an auxiliary process by means of which the NS access by TC instances is restricted to the interactions of their concern. More precisely, the N-DATA and N-DATA-ACKNOWLEDGE service primitives are executed by the class1 TC instances whenever the Receipt Confirmation NS option is made use of.

The internal gate *r* is introduced in this definition, where the release of a TC is indicated (see 14.3.2): the associated information refers also to the frozen references procedure (see clause 15).

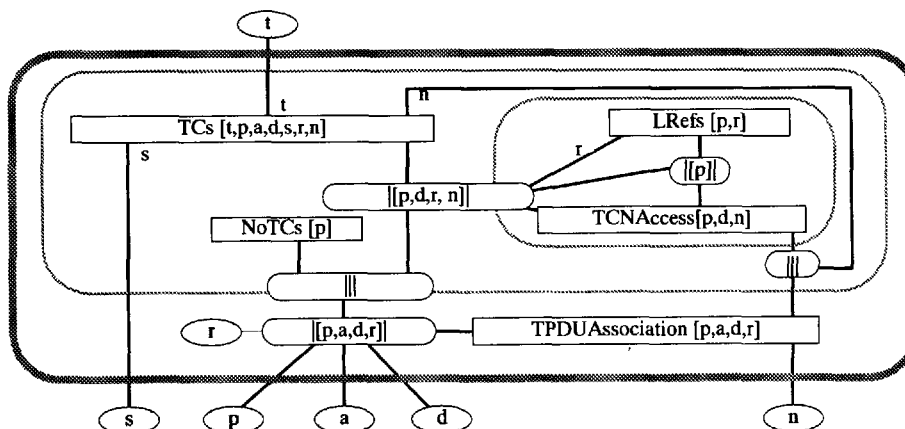


Figure 4 - Structure of process TSTP

```

process TSTP[t, p, a, d, s, n](tpeo : TPEOptions) : noexit:=
hide r
in
( ( TCs [t, p, a, d, s, r, n] (tpeo)
  |[p, d, r, n]|
  LRefs [p, r] |[p]| TCNAccess [p, d, n]
)
  |||
  NoTC [p]
)

```

```

|[p, a, d, r]|
TPDUAssociation [p, a, d, r] (Class4 IsIn Classes(tpeo))
endproc (* TSTP *)
process TCs[t, p, a, d, s, r, n](tpeo : TPEOptions) : noexit:=
TC [t, p, a, d, s, r, n] (tpeo)
|||
TCs [t, p, a, d, s, r, n] (tpeo)
endproc (* TCs *)
(*

```

### 13.2.3 Usage of network connections

The constraints on usage of NCs can be formulated locally to each NC. *TPNS* therefore consists of an unbounded number of parallel instances of process *NC* (see 17.1).

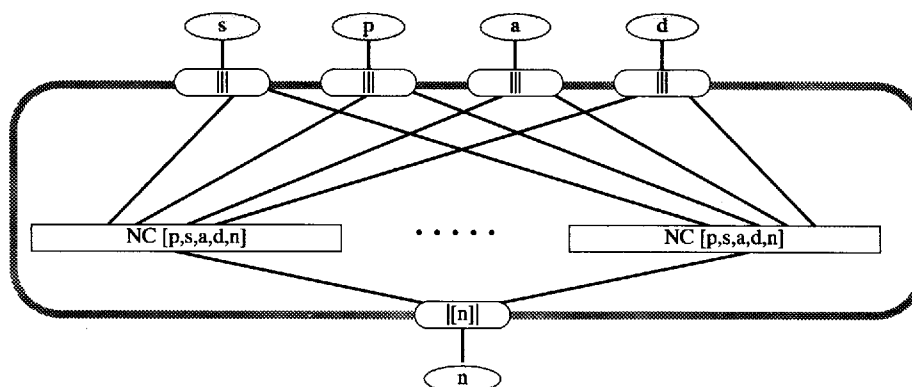


Figure 5 - Structure of process TPNS

```

process TPNS[p, a, d, s, n] : noexit:=
NC [p, a, d, s, n]
|||
TPNS [p, a, d, s, n]
endproc (* TPNS *)
*)

```

## 14 Event Structures

### 14.1 General

The event structures are informally presented in 14.2 that relate to the service gates and to the internal gates introduced in 13.1

### 14.2 Interactions at the service boundaries

The event structure at gate *t* is the same as in the TS FD, except that a more refined construction of T-DATA primitives is available here (see 8.2 for *TAddress*, see clause A.2 for *TCEI*, and see clause 12 for the refinement of TSP):

*t* ?*ta*:TAddress ?*tcei*:TCEI ?*tsp*:TSP

The event structure at gate *n* needs no refinement, as segmenting and flow control can be correctly represented even if atomic execution of N-DATA primitives is assumed (see 8.2 for the definitions of *NAddress*, *NCEI*, and see clause B.2 for *NSP*):

*n* ?*na*:NAddress ?*ncai*:NCEI ?*nsp*:NSP

### 14.3 Interactions at internal gates

#### 14.3.1 TPDU transfer

The events at gate *p* have the following structure:

*p* ?*tr*:TPId ?*ni*:NCId ?*cl*:Class ?*d*:Dir ?*c*:Copy ?*tpdu*:ETPDU ?*err*:TPErr

where

## ISO/IEC TR 10024 : 1992(E)

*tr:TPId* (see 15.2) is the protocol identification (see 15.1) of the TC with which the TPDU is associated,

*ni:NCId* (see 17.2) identifies the NC on which the TPDU is transferred by local NSAP address, Remote NSAP Address and NC Endpoint Identifier

*cl:Class* (see 8.3.1) is the protocol class in which the TPDU is handled,

*d:Dir* (see below) tells whether the TPDU is to be transmitted or was received, on a NC,

*c:Copy* (see below) tells whether or not the TPDU is considered a duplicate. Only TPDU's that, possibly, have been retransmitted are considered as being duplicates,

*tpdu:ETPDU* (see clause 40) is the TPDU to be transferred; this value is represented in its encoded form for technical reasons that derive from the procedure for handling protocol errors (see clause 30 for details),

*err:TPErr* significant for received TPDU's, qualifies protocol errors; a few error levels are defined (see clause 39).

\*)

```

type Direction
is TwoTuplet renamedby
sortnames
  Dir for Tuplet
opnnames
  Send for TheOne
  Recv for TheOther
endtype (* Direction *)

type Copy
is TwoTuplet renamedby
sortnames
  Copy for Tuplet
opnnames
  New for TheOne
  Dupl for TheOther
endtype (* Copy *)

```

(\*)

## 14.3.2 Transport connection release

As anticipated in 13.2.2, events at *r* indicate TC release. More precisely, any interaction at *r* consists of:

- the indication on whether or not the reference is to be frozen, significant only if a local reference was allocated to the TC,
- the identification (see 15.2) of the TC that is released,
- The reason why a connection is released. This information is needed for the release procedures (see clause 22).

The events at gate *r* have the following structure:

*r* ?f:RefReuse ?tr:TPId ?w:RelReason

where *RefReuse* is either *Free* or *Freeze*.

The values of *RelReason* represent the following reasons for releasing a TC:

|                |                                                                                                                                                 |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>Normal</i>  | Release on request of the User. This value is generated by <i>TCRelS</i> (see 22.2).                                                            |
| <i>Error</i>   | Release caused by occurrence of an invalid TPDU or protocol error. This value is generated by <i>TCE</i> (see 16.1).                            |
| <i>NoReass</i> | Release caused by the decision not to reassign a TC to a new NC. This value is generated by <i>TCA</i> (see 25.2.1).                            |
| <i>NoResyn</i> | Release caused by the decision not to resynchronize after occurrence of an N-RESET primitive. This value is generated by <i>TCE</i> (see 16.1). |

When the TC is a class 2 connection, the values *NoReass* and *NoResyn* cause release according to subclause 10.1 of the protocol.

\*)

```

type RefReuse
is TwoTuplet renamedby
sortnames
  RefReuse for Tuplet
opnnames
  Free for TheOne

```

```

    Freeze for TheOther
endtype (* RefReuse *)
type ReleaseInfo
is FourTuplet renamedby
sortnames
    RelReason for Tuplet
opnnames
    Normal for TheOne
    Error for TheOther
    NoReass for TheThird
    NoResyn for TheFourth
endtype (* ReleaseInfo *)
(*)

```

### 14.3.3 Assignment to and deassignment from network connection

As mentioned above, the internal gates *a* and *d* facilitate the formal representation of (re)assignment to an existing NC. The related event structures reflect this purpose:

The information passed at *a* consists of TC and NC identifications, and of NC ownership, NC parameters (viz. addresses and options) as determined upon successful NC establishment, and NC multiplexing possibility (see 17.1 and clause 32);

The event structure at *a* is as follows:

```

    a ?tr:TPId ?ni:NId ?own:NCO ?ncp:NCPAr ?mx:MuxMode

```

The need of three values to represent the status of a NC w.r.t. multiplexing arises from the requirement that multiplexing shall not commence until allowed by class selection (see subclause 6.5.4.j of the protocol). This requirement is described in clause 32 below.

The minimality of the structure of *NCPParameters* (viz., no representation of QOS) is justified by the implementation-dependent character of the relationship between QOS of the TC provided and QOS of the NC(s) which the TC is assigned to (see note 1 in subclause 6.1.1.3 of the protocol). This relationship is not specified in this formal description. (see 8.4 for *TwoTuplet*, and 41.2 for *ThreeTuplet*).

The sort *NCPAR* is defined in *BasicNCPParameters* and is enriched with Boolean functions that enables the selection of the NC parameters from a N-CONNECT Service Primitive.

```

type NCOwnership
is TwoTuplet renamedby
sortnames
    NCO for Tuplet
opnnames
    Owner for TheOne
    NonOwner for TheOther
endtype (* NCOwnership *)
type NCMultiplexingMode
is ThreeTuplet renamedby
sortnames
    MuxMode for Tuplet
opnnames
    Mux for TheOne
    NoMux for TheOther
    WaitMux for TheThird
endtype (* NCMultiplexingMode *)
type BasicNCPParameters
is Pair actualizedby NetworkAddress, NSOptions using
sortnames
    NAddress for Element
    NSO for Element2
    Bool for FBool
    NCPAr for Pair
opnnames
    RNA for First
    NSO for Second
    NCPAr for Pair
endtype (* BasicNCPParameters *)

```

\*)

## ISO/IEC TR 10024 : 1992(E)

```

type NParameters
is BasicNParameters, NetworkServicePrimitive
opns
  _IsNParOf_ : NPar, NSP -> Bool
eqns
forall
  a, a1, a2 : NAddress, o, o1 : NSO, q1 : NQOS, n : NSP, p : NPAR
ofsort Bool
  NPar(a, o) IsNParOf NCONreq(a1, a2, o1, q1) = a eq a2 and (o eq o1);
  NPar(a, o) IsNParOf NCONind(a1, a2, o1, q1) = a eq a2 and (o eq o1);
  NPar(a, o) IsNParOf NCONconf(a1, o1, q1) = a eq a1 and (o eq o1);
  NPar(a, o) IsNParOf NCONresp(a1, o1, q1) = a eq a1 and (o eq o1);
  not(IsNCON(n)) => p IsNParOf n = false;
endtype (* NParameters *)

```

(\* \_\_\_\_\_ \*)  
 The information passed at *d* consists of TC and NC identifications and of a qualifier indicating whether or not the NC is to be disconnected.

*TCDDis* NC is to be deassigned from all TCs assigned to it and is to be disconnected.

*TCDDis* deassignment initiative comes from a non class 0 TC.

The structure at the *d* gate is as follows.

*d* ?tr:TPId ?ni:NId ?w:Deassign

where *TPId* is defined in 15.2, *NId* in 17.2.

```

(* _____ *)
type TCNCDeassignment
is ThreeTuplet renamedby
sortnames
  Deassign for Tuplet
opnnames
  TCDDis for TheOne
  TCDDis for TheOther
  NCD for TheThird
endtype (* TCNCDeassignment *)

```

#### 14.3.4 Network reset

The reason for the internal gate *s* consists in transferring the information about occurrences of a reset of a NC to the TC instances (see 16.1) that are assigned to that NC. The structure of events at *s* is then simply (see 8.2 for *NId*):

*s* ?ni:NId

A few other gates will be introduced in the following clauses. They are explained when used.

## 15 Identification of transport connections

### 15.1 General

At the TS boundary each TC is uniquely identified by the address *ta* of the TSAP where the TC is provided and by a TC endpoint identifier *tcei*: this, within the given TSAP, is uniquely allocated to the TC for its lifetime at the service boundary. The pair (*ta*, *tcei*) is referred to as the service identification of the TC. A similar identification mechanism is defined at the NS boundary.

The protocol specifies an additional mechanism for TC identification. This is necessary because the lifetime of the TC at the TSAP does not coincide with its lifetime within the transport entity. This mechanism is based on the exchange of references between the transport entities. The formal representation of this mechanism in this Technical Report (see 15.2) is referred to as the protocol identification of the TC.

In most cases the protocol identification is determined by the local reference assigned to the TC. In some cases, however, the local reference may well be unavailable (e.g., connection refusal). In such cases the protocol identification is determined by the remote reference, together with the remote network address (see 15.2 for further details).

The structure chosen for the protocol identification of the TC is justified by the following wish: specifying the allocation of a free (i.e. neither in use nor frozen) local reference to a TC only at the stage when - according to the protocol - such an allocation is strictly necessary, viz. when a new CR or CC TPDU is to be sent.

Although in some implementations the implementor may choose to allocate the new local reference at some earlier stage, in other implementations the allocation may occur only when it becomes necessary according to the protocol. Catering for the first case only, therefore, would definitely introduce a certain amount of overspecification.

## 15.2 Data definitions

*TPReference* and *TPReferences* are basic constructions, which respectively define the protocol references and finite sets of them. *OctetPair* is defined in 41.2.

The data type *TCProtocolId* is designed to comprise the different identification mechanisms defined at different stages of a TC. A basic construction is presented first, where some (the most frequently used) values of sort *TPId* result from application of the function *Local* to a reference. Further values of this sort are yielded by application of:

- a) *TIdRef* to a pair *TId* x *Ref* (see Note 1),
- b) *Remote* to a pair *NAddress* x *Ref* (see Note 2),
- c) *Undefined*, which is a constant (see Note 3).

The functions *Ref* and *Qual* can be applied to a *tr* of sort *TPId*. *Ref* returns the reference with which *tr* is built, but if *tr* is *Undefined* then *Ref* returns *ZeroRef*. *Qual* qualifies the reference with which *tr* is built by returning either *Local* or *Remote*, but if *tr* is *Undefined* then *Qual* yields *Undefined*.

The *TCProtocolId* type definition enriches the basic construction with the functions *LocalSrcRef* and *RemoteSrcRef*, which are made use of in 16.3. The type *ETPDU* is defined in clause 40.

Finally, the definition *LocalReferenceAllocation* provides a boolean function that characterizes those TPDU transfers that require allocation of a free (i.e. neither in use nor frozen) reference. This function is used in 15.3.3 and 16.3.

NOTE – 1 Sending a new (c = New) CR TPDU is the only case where also the service identification of the TC is needed at p. This is so because the assignment of the TC to a NC does not necessarily require the allocation of a local reference to the TC. It is necessary, however, that the CR TPDU is sent on the NC to which the TC was assigned. Such an assignment, at this stage, can only be represented as a *TId* x *NId* association. A local reference (neither in use nor frozen) is also selected at this stage. Values passed at the a gate are always constructed using the *TIdRef* constructor. Passing both the *TId* and the *Ref* information in this 'send new CR' exchange allows to 'switch' the representation of the assignment - viz. from the *TId* x *NId* association to the *Ref* x *NId* association - but, at the same time, to keep the same assignment of the TC to the NC. Therefore the value of *tr* in this situation consists of a pair *TId* x *Ref*.

NOTE – 2 In the connection establishment phase the responder of a TC identifies that TC by means of the remote reference carried in the received CR TPDU. Such an identification, however, is not sufficient because another peer entity (accessing therefore another remote NSAP) can initiate a different TC using an identical reference: references are unique within the scope of a transport entity only.

This is the reason why also a network address is needed to distinguish such TCs. This address is the remote NSAP address of the NC on which the TPDU is received; this address and the remote reference together suffice to identify the TC (see first sentence in subclause 6.9.1.4.2 of the protocol).

It is to be noted that such an identification also suffices to cater for the appropriate handling of protocol errors that are generated by a peer initiator entity that starts splitting before receipt of a CC TPDU, but only if the peer entity does so by using the same NSAP (see the first Note in subclause 6.5.4 and Note 2 in subclause 6.23.2 of the protocol). In the responder case, the value of *tr* - until a CC TPDU is sent - is yielded by application of *Remote* to a pair <*NAddress* x *Ref*>.

NOTE – 3 TPDU's which are not associated with any TC are handled by process *NoTC* (see 31.2). This applies both to received TPDU's that cannot be associated, according to subclause 6.9.1.4.2 of the protocol, and to TPDU's sent by this process. The *Undefined* value of *tr* is used when passing TPDU's that are not associated to any TC.

\*)

```

type TPreference
is BasicTPreference
opns
  ZeroRef          :                               -> Ref
eqns
ofsort Ref
  ZeroRef = Ref(Octet(0, 0, 0, 0, 0, 0, 0, 0), Octet(0, 0, 0, 0, 0, 0, 0, 0));
endtype (* TPreference *)

type TPReferences
is Set actualizedby TPreference using
sortnames
  Ref for Element
  Bool for FBool
  Refs for Set
endtype (* TPReferences *)

type BasicTPreference
is OctetPair renamedby
sortnames
  Ref for OctetPair
opnnames

```

## ISO/IEC TR 10024 : 1992(E)

```

    Ref for Pair
endtype (* BasicTPReference *)
type BasicTCPProtocolId
is TPReference, TPReferenceQualifier, TCEIdentification, NetworkAddress
sorts
    TPId
opns
    Local          : Ref          -> TPId
    TIdRef         : TId, Ref     -> TPId
    Remote         : NAddress, Ref -> TPId
    Undefined      :              -> TPId
    Ref            : TPId         -> Ref
    Qual           : TPId         -> Qual
    _eq_, _ne_     : TPId, TPId   -> Bool
eqns
forall
    r : Ref, ti : TId, na : NAddress, tr, tr1 : TPId
ofsort Ref
    Ref(Local(r)) = r;
    Ref(TIdRef(ti, r)) = r;
    Ref(Remote(na, r)) = r;
    Ref(Undefined) = ZeroRef;
ofsort Qual
    Qual(Local(r)) = Local;
    Qual(TIdRef(ti, r)) = Local;
    Qual(Remote(na, r)) = Remote;
    Qual(Undefined) = Undefined;
ofsort Bool
    tr eq tr1 = Ref(tr) eq Ref(tr1) and (Qual(tr) eq Qual(tr1));
    tr ne tr1 = not(tr eq tr1);
endtype (* BasicTCPProtocolId *)
type TPReferenceQualifier
is ThreeTuplet renamedby
sortnames
    Qual for Tuplet
opnnames
    Local for TheOne
    Remote for TheOther
    Undefined for TheThird
endtype (* TPReferenceQualifier *)
type TCPProtocolId
is BasicTCPProtocolId, TPDUEncoding
opns
    LocalSrcRef, RemoteSrcRef : TPId, ETPDU          -> Bool
eqns
forall
    tr : TPId, tpdu : ETPDU
ofsort Bool
    LocalSrcRef(tr, tpdu) = Qual(tr) eq Local and (Ref(tr) eq SrcRef(tpdu));
    RemoteSrcRef(tr, tpdu) = Qual(tr) eq Remote and (Ref(tr) eq SrcRef(tpdu));
endtype (* TCPProtocolId *)
type LocalReferenceAllocation
is Direction, Copy, TPDUEncoding
opns
    AssignLocalRef          : Dir, Copy, ETPDU          -> Bool
eqns
forall
    d : Dir, c : Copy, tpdu : ETPDU
ofsort Bool
    AssignLocalRef(d, c, tpdu)
      = d eq Send and (c eq New) and (IsCR(tpdu) or IsCC(tpdu));
endtype (* LocalReferenceAllocation *)
  (*

```

## 15.3 Process definitions

### 15.3.1 General

The following definitions present the requirement of unique identification of transport connections by means of local references, as well as the frozen references procedure. These requirements are specified by process *LRefs*. Further constraints on TC identification are defined locally to each TC (see 16.3). The constraints on the allocation states of each individual reference are described separately (see 15.3.3).

### 15.3.2 Global constraints

Process *LRefs* comprises the following subprocesses:

- a) *LocalReferences*: this process consists of a number of parallel instances of process *FreeReference*, each parameterized with a distinct, non-zero local reference. The number of references supported depends on the implementation, and is not specified,
- b) *RunNonLocalReferences*(see 41.1): by means of this auxiliary definition only the interactions of concern are constrained.

---

```

process LRefs[p, r] : noexit:=
choice lrs : Refs
[]
[lrs ne {} and (ZeroRef NotIn lrs)] ->
  i ; ( LocalReferences [p, r] (lrs)
      |||
      RunNonLocalReferences [p])
endproc (* LRefs *)

process LocalReferences[p, r](lrs : Refs) : noexit:=
choice lr : Ref
[]
[lr IsIn lrs] ->
  ( FreeReference [p, r] (lr)
    |||
    LocalReferences [p, r] (Remove(lr, lrs)))
endproc (* LocalReferences *)
  
```

---

### 15.3.3 Constraints for a single reference

*FreeReference*, *BoundReference* and *FrozenReference* represent the possible states of a local reference w.r.t. its allocation to a TC. Every reference is initially free. A local reference becomes bound, i.e. allocated to a TC, upon occurrence of the first event at p that carries it as a local reference identification of the TC.

A bound local reference is exchanged in every interaction at p that has the *Local* qualifier of the TC identification and that does not invite transmission of a non-duplicate CR or CC TPDU to the peer entity.

A bound reference may return to the initial state only if the TC is released. This condition is satisfied upon and after the occurrence of an event at r that carries this reference: the reference is then freed either immediately or after some time - during which the reference is frozen. This is represented by the internal event in the definition of *FrozenReference*.

NOTE – The frozen reference procedure relates to classes 1, 3 and 4, but implementations may follow this procedure also in other classes (see subclause 6.18 of the protocol). The conditions under which the reference may and/or must be frozen are specified in clause 22, as far as they depend on the sequence of TPDU's that has led to release of the TC, and in clause 25 as far as they depend on the outcome of the reassignment and resynchronization procedures.

---

```

process FreeReference[p, r](lr : Ref) : noexit:=
p?tr : TPId ?nci : NCId ?cl : Class ?d : Dir ?c : Copy ?tpdu : ETPDU ?err : TPErr
[Qual(tr) eq Local and (Ref(tr) eq lr)] ;
BoundReference [p, r] (Local(lr))
endproc (* FreeReference *)

process BoundReference[p, r](tr : TPId) : noexit:=
UseBoundReference [p] (tr)
[>
r!Free !tr ?w : RelReason ; FreeReference [p, r] (Ref(tr))
[]
r!Freeze !tr ?w : RelReason ; FrozenReference [p, r] (Ref(tr))
endproc (* BoundReference *)
  
```

---

## ISO/IEC TR 10024 : 1992(E)

```

process UseBoundReference[p](tr : TPId) : noexit:=
p!tr ?nci : NCId ?cl : Class ?d : Dir ?c : Copy ?tpdu : ETPDU ?err : TPErr
  [not(AssignLocalRef(d, c, tpdu))] ;
UseBoundReference [p] (tr)
endproc (* UseBoundReference *)

process FrozenReference[p, r](lr : Ref) : noexit:=
i ; FreeReference [p, r] (lr)
endproc (* FrozenReference *)

(*

```

## 16 Provision of a transport connection

### 16.1 General

The following classification of constraints on provision of a TC is reflected in the definition of process *TC* (see Figure 6):

- a) service constraints, i.e. constraints at *t* only, which are specified by *TCEP* (see clause A.2),
- b) protocol constraints, i.e. constraints that relate the events at *t* to the other interactions of the transport entity (w.r.t. provision of the given TC).

Protocol constraints are further classified as follows:

- a) TC identification, specified by *TCId* (see 16.3), which relates the service identification of a TC with its protocol identification (see 15.1),
- b) relationship between TSPs and TPDU, specified by *TSPTPDU* (see clause 21),
- c) normal exchange of TPDU between the two peer entities, specified by *TPDUFlowProcedures* independently of the events at *t* and of error recovery procedures,
- d) (re)assignment of the TC to one (or more) NC(s), specified by *TCA* (see 25.2): this process specifies also the TC release that follows expiration of the reassignment/resynchronization timers,
- e) TC release, specified by *TCRelease* (see clause 22),
- f) exception handling, specified by *TCE*: this comprises error recovery procedures (in classes 1, 3, 4), specified by *NSError-Recovery* (see clause 27), and treatment of protocol errors, specified by *ProtocolErrorHandling* (see clause 39).

The procedures relating to normal flow of TPDU between the two peer entities are described by the following processes:

- a) *SupportedOptions* (see 20.1),
- b) *TPNegotiations* (see 20.2),
- c) *DataTPDUNumbering* (see 23.1),
- d) *ExplicitFlowControl* (see clause 24),
- e) *DTRetransmission* (see 28.7),
- f) *Checksum* (see clause 34)
- g) *TPDUEncoding* (see clause 40).

NOTE – The TC components accessing *r* are terminating processes, while the others invoke non-terminating ones, but may terminate at any time - in the TC context, because of the **exit** [>] construct.

The internal gate *k* introduced by the definition of TC serves the purposes of separating concerns in the description of error recovery mechanisms, particularly relating to retention of TPDU until acknowledgement and retransmission. The structure of events at *k* is as follows:

$k \text{ ?s:TPDUSubsort ?nd:TPDUStatus}$

where  $s \in \{CR, CC, DR, DT, ED\}$  (see 19.3) and *nd* is a pair of natural numbers. An event at *k* carries information about the availability of one or more TPDU of Subsort *s* for transmission or retransmission. More precisely: the first component of *nd* tells of such

TPDUs that may be transmitted, with none of them ever transmitted before, whilst the second component of *nd* tells the number of them that are retained for possible retransmission. The functions *New* and *Dupl* on *TPDUStatus* yield these components.

Events at *k* concern retention and retransmission (see clause 28), reassignment after failure (only because the existence of a - possibly retained - DR TPDU may affect the decision of whether or not to reassign, see clause 26 and clause 27), resynchronization (see clause 30), and - during resynchronization - transmission of ED TPDU, TPDU numbering, explicit flow control.

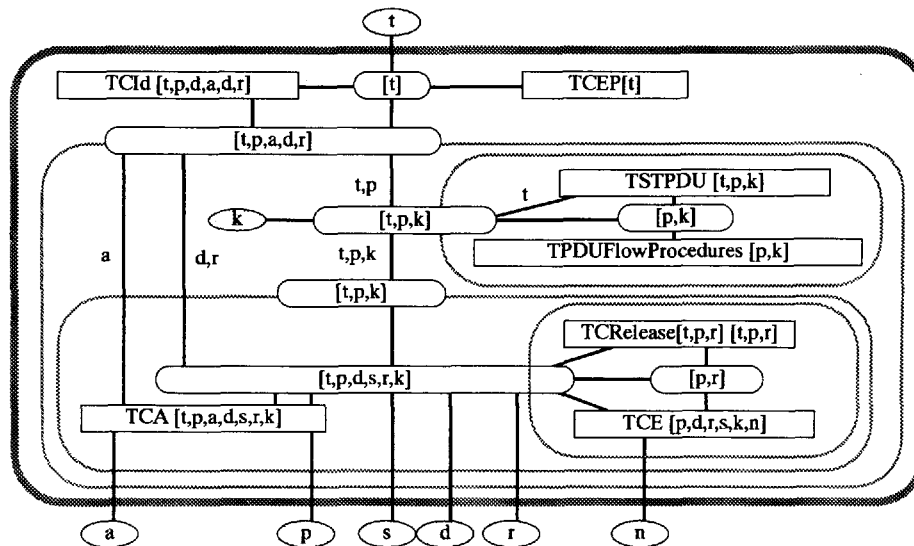


Figure 6 - Structure of process TC

```

process TC[t, p, a, d, s, r, n](tpeo : TPEOptions) : exit :=
  TCEP [t] (TSCallingRole) [] TCEP [t] (TSCalledRole)
  |[t]|
  (TCId [t, p, a, d, r] [> exit ]
  |[t, p, a, d, r]|
  (hide k
    in
      ( TSPTPDU [t, p, k] [> exit
        |[p, k]|
        TPDUFlowProcedures [p, k] (tpeo) [> exit
      )
      |[t, p, k]|
      TCA [t, p, a, d, s, r, k] (tpeo)
      |[t, p, d, s, r, k]|
      ( TCRelease [t, p, r]
        |[p, r]|
        TCE [p, d, r, s, k, n] [> exit
      )
    )
  )
endproc (* TC *)

type TPDUStatus
is Pair actualizedby NaturalNumber using
sortnames
  Nat for Element
  Nat for Element2
  Bool for FBool
  TPDUStatus for Pair
opnnames
  New for First
  Dupl for Second
endtype (* TPDUStatus *)
  
```

\*)

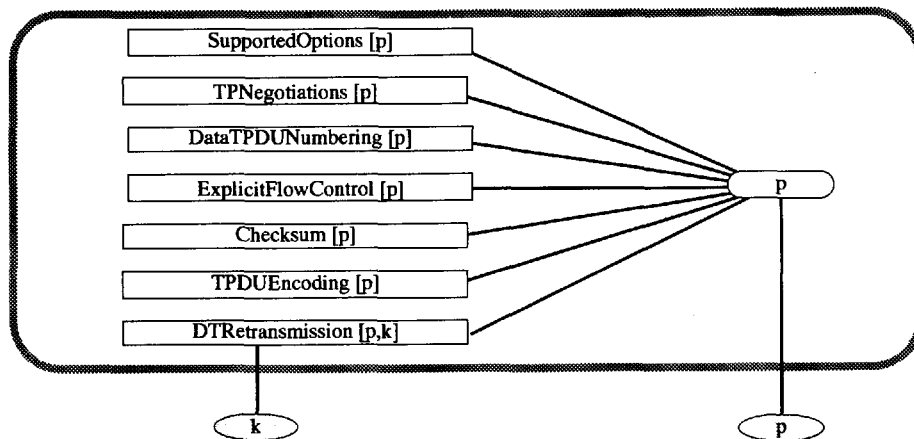


Figure 7 - Structure of process TPDUFLOWPROCEDURES

```

*)
process TPDUFLOWPROCEDURES[p, k](tpeo : TPEOptions) : noexit:=
(SupportedOptions [p] (tpeo)
  ||
  TPNegotiations [p]
  ||
  TPDUNumbering [p]
  ||
  ExplicitFlowControl [p]
  ||
  Checksum [p]
  ||
  TPDUEncoding [p]
)
|[p]| DTRetransmission [p, k]
endproc (* TPDUFLOWPROCEDURES *)
process TCE[p, d, r, s, k, n] : noexit:=
NSErrorRecovery [p, d, s, k, n]
|[p, s, d]|
ProtocolErrorHandling [p, s, d, r]
endproc (* TCE *)
(*)

```

## 16.2 Service constraints

TCEP (see clause A.2) prescribes usage of the same service identification, and the TS ordering requirements on TSPs relating to a single TCEP, for the TC lifetime at that TCEP.

## 16.3 Identification of the transport connection

TCId consists of five subprocesses, viz.

- a) *TCIdByRef*, which specifies the protocol identification of the TC at gate *p*: if the entity is the initiator it may only use one selected *lr* for the lifetime of a TC; if the entity is the responder the remote reference is first employed, together with the remote NSAP address, to identify the connection, until a CC TPDU is sent and therefore a local reference is assigned (see 15.2, Note 2),

*TCIdTP*, which relates the protocol identification of the TC to its service identification whenever the transport entity is the initiator of the TC (see 15.2 Note 1),

- b) three instances of *TCIdAt\_x*, that specify the TC identification at gates *a*, *d* and *r* respectively.

```

*)
process TCId[t, p, a, d, r] : noexit:=
TCIdByRef [p]
|[p]|
TCIdTP [t, p]

```

```

| [t, p] |
TCId_at_a [t, p, a]
| [t, p] |
TCId_at_d [t, p, d]
| [t, p] |
TCId_at_r [t, p, r]
endproc (* TCId *)

process TCIdByRef[p] : noexit:=
p?tr : TPId ?nci : NCId ?cl : Class !Send !New ?tpdu : ETPDU ?err : TPErr
[LocalSRCRef(tr, tpdu)] ;
TCLocalRef [p] (Local(Ref(tr)))
[]
p?tr : TPId ?nci : NCId ?cl : Class !Recv ?c : Copy ?tpdu : ETPDU ?err : TPErr
[RemoteSRCRef(tr, tpdu)] ;
TCRemoteRef [p] (tr)
endproc (* TCIdByRef *)

process TCLocalRef[p](lr : TPId) : noexit:=
p!lr ?nci : NCId ?cl : Class ?d : Dir ?c : Copy ?tpdu : ETPDU ?err : TPErr ;
TCLocalRef [p] (lr)
endproc (* TCLocalRef *)

process TCRemoteRef[p](rr : TPId) : noexit:=
p!rr ?nci : NCId ?cl : Class ?d : Dir ?c : Copy ?tpdu : ETPDU ?err : TPErr
[not(AssignLocalRef(d, c, tpdu))] ;
TCRemoteRef [p] (rr)
[]
p?lr : TPId ?nci : NCId ?cl : Class !Send !New ?tpdu : ETPDU ?err : TPErr
[AssignLocalRef(Send, New, tpdu) and LocalSRCRef(lr, tpdu)] ;
TCLocalRef [p] (lr)
endproc (* TCRemoteRef *)

process TCIdTP[t, p] : noexit:=
t?ta : TAddress ?tcei : TCEI ?tsp : TSP ;
(Run_at_t [t]
| | |
(choice lr : Ref
[]
p!TidRef(Tid(ta, tcei), lr) ?nci : NCId ?cl : Class ?d : Dir ?c : Copy
?tpdu : ETPDU ?err : TPErr ;
Run_at_p [p]))
[]
p?Tid : TPId ?nci : NCId ?cl : Class ?d : Dir ?c : Copy ?tpdu : ETPDU ?err : TPErr ;
(Run_at_t [t] | | | Run_at_p [p])
endproc (* TCIdTP *)

process TCId_at_a[t, p, a] : noexit:=
t?ta : TAddress ?tcei : TCEI ?tsp : TSP ;
(Run_at_t [t] | | | TCIdStarted_at_a [p, a] (TidRef(Tid(ta, tcei), ZeroRef)))
[]
p?tr : TPId ?nci : NCId ?cl : Class ?d : Dir ?c : Copy ?tpdu : ETPDU ?err : TPErr ;
(Run_at_t [t] | | | TCIdStarted_at_a [p, a] (tr))
endproc (* TCId_at_a *)

process TCId_at_d[t, p, d] : noexit:=
t?ta : TAddress ?tcei : TCEI ?tsp : TSP ;
(Run_at_t [t] | | | TCIdStarted_at_d [p, d] (TidRef(Tid(ta, tcei), ZeroRef)))
[]
p?tr : TPId ?nci : NCId ?cl : Class ?d : Dir ?c : Copy ?tpdu : ETPDU ?err : TPErr ;
(Run_at_t [t] | | | TCIdStarted_at_d [p, d] (tr))
endproc (* TCId_at_d *)

process TCId_at_r[t, p, r] : noexit:=
t?ta : TAddress ?tcei : TCEI ?tsp : TSP ;
(Run_at_t [t] | | | TCIdStarted_at_r [p, r] (TidRef(Tid(ta, tcei), ZeroRef)))
[]
p?tr : TPId ?nci : NCId ?cl : Class ?d : Dir ?c : Copy ?tpdu : ETPDU ?err : TPErr ;
(Run_at_t [t] | | | TCIdStarted_at_r [p, r] (tr))
endproc (* TCId_at_r *)

process TCIdStarted_at_a[p, a](tr : TPId) : noexit:=
ConstantTCId_at_a [a] (tr) | | | RunButAssignLocalRef [p]

```

## ISO/IEC TR 10024 : 1992(E)

```

[>
p?tr : TPId ?nci : NCId ?cl : Class !Send !New ?tpdu : ETPDU ?err : TPErr
  [AssignLocalRef(Send, New, tpdu)] ;
  (ConstantTCId_at_a [a] (Local(Ref(tr))) ||| Run_at_p [p])
endproc (* TCIdStarted_at_a *)

process TCIdStarted_at_d[p, d](tr : TPId) : noexit:=
ConstantTCId_at_d [d] (tr) ||| RunButAssignLocalRef [p]
[>
p?tr : TPId ?nci : NCId ?cl : Class !Send !New ?tpdu : ETPDU ?err : TPErr
  [AssignLocalRef(Send, New, tpdu)] ;
  (ConstantTCId_at_d [d] (Local(Ref(tr))) ||| Run_at_p [p])
endproc (* TCIdStarted_at_d *)

process TCIdStarted_at_r[p, r](tr : TPId) : noexit:=
ConstantTCId_at_r [r] (tr) ||| RunButAssignLocalRef [p]
[>
p?tr : TPId ?nci : NCId ?cl : Class !Send !New ?tpdu : ETPDU ?err : TPErr
  [AssignLocalRef(Send, New, tpdu)] ;
  (ConstantTCId_at_r [r] (Local(Ref(tr))) ||| Run_at_p [p])
endproc (* TCIdStarted_at_r *)

process ConstantTCId_at_a[a](tr : TPId) : noexit:=
a!tr ?ni : NId ?own : NCO ?ncp : NCPAr ?mx : MuxMode ; ConstantTCId_at_a [a] (tr)
endproc (* ConstantTCId_at_a *)

process ConstantTCId_at_d[d](tr : TPId) : noexit:=
d!tr ?ni : NId ?w : Deassign ; ConstantTCId_at_d [d] (tr)
endproc (* ConstantTCId_at_d *)

process ConstantTCId_at_r[r](tr : TPId) : noexit:=
r?f : RefReUse !tr ?w : RelReason ; ConstantTCId_at_r [r] (tr)
endproc (* ConstantTCId_at_r *)
(*

```

## 17 Usage of a network connection

### 17.1 General

The following functional decomposition of NC is presented:

- NCEP*: service identification of the NC (see clause B.2),
- NCId*: protocol identification of the NC (see 17.2),
- NCH*: transfer of TPDU's (see 17.3),
- NCA*: (de)assignment of TC(s) (see 25.3),
- NCM*: multiplexing constraints (see 32.2.1).
- NCE*: Signalling of NS errors to TCs, possibly under multiplexing (see 32.2.2).

*NCH*, *NCA*, and *NCE* do not synchronize at *n* as they engage in different service primitives. In particular:

*NCH* in N-DATA, N-EXP-DATA

*NCA* in N-CONNECT, N-DISCONNECT

*NCE* in N-RESET.

*NCM* describes the decision(s) of whether or not to multiplex the NC. This process also describes the constraints on (possibly temporary) selection of protocol class that relate to multiplexing. The internal gate *m* is introduced to let the concerned NC components, viz. *NCA* and *NCM*, have the same view about NC multiplexing. The structure of events at *m* is therefore:

*m* ?mx:MuxMode [mx ne WaitMux]

with only those values of sort *MuxMode* that represent a final decision on whether or not to multiplex the NC.

NOTE – Usage of N-DATA-ACKNOWLEDGE primitives is described as part of *NSErrorRecovery*, a subprocess of *TC* (see clause 16 and clause 27).

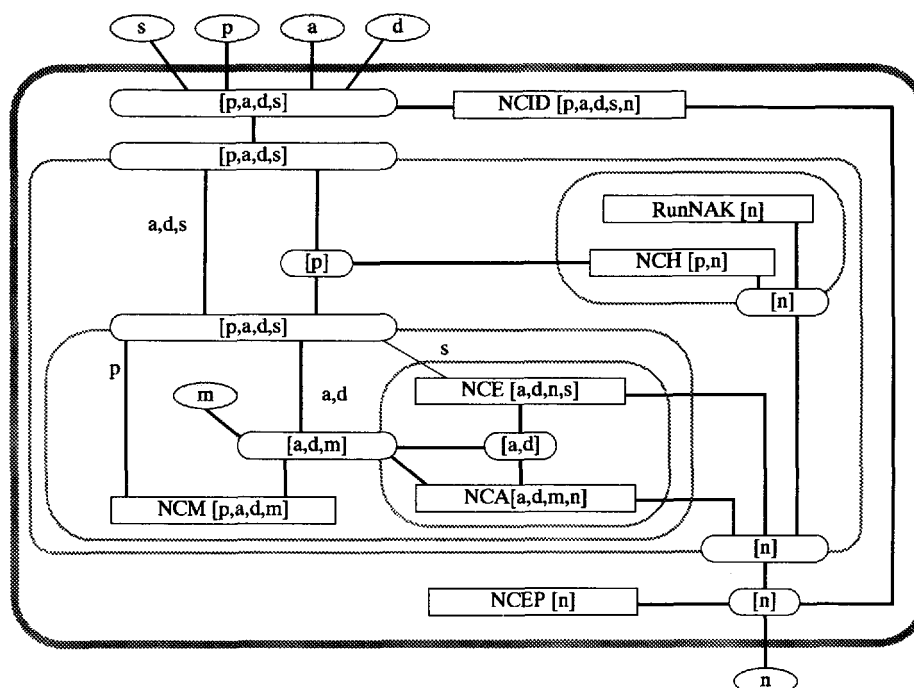


Figure 8 - Structure of process NC

```

process NC[p, a, d, s, n] : exit:=
(NCEP [n] (NSCallingRole) [] NCEP [n] (NSCalledRole)
|[n]| (NCId [p, a, d, s, n] [> exit]))
||
(NCH [p, n] ||| RunNAK [n] [> exit])
|[p]|
(hide m
in
(NCA [a, d, m, n] |[a, d]| (NCE [a, d, n, s] [> exit]))
|[a, d, m]| (NCM [p, a, d, m] [> exit]))
endproc (* NC *)

```

## 17.2 Identification of the network connection

*NCId* specifies usage of the same NC identification at all of the internal gates where the NC identification is passed. The identification is established in the first interaction at the NS boundary that starts the NC (see clause B.2 for the function *NId*). At the *a*, *d*, *s* and *n* gates, the identification consists of a pair <NSAP address, NC Endpoint Identifier>. This pair is, extended with the remote NSAP address, also used at the *p* gate.

```

process NCId[p, a, d, s, n] : noexit:=
n?NA : NAddress ?ncei : NCEI ?ncr : NSP ;
(choice ncp : NCPPar
[]
[ncp IsNCPParOf ncr] ->
(let ni : NId = NId(NA, ncei)
in
Run_at_n [n]
|||
NId_at_p [p] (NCId(ni, RNA(ncp)))
|||
NId_at_a [a] (ni)
|||
NId_at_d [d] (ni)
|||

```

## ISO/IEC TR 10024 : 1992(E)

```

    NId_at_s [s] (ni)))
endproc (* NCId *)
process NId_at_a[a](ni : NId) : noexit:=
a?tr : TPId !ni ?own : NCO ?ncp : NCPAr ?mx : MuxMode ; NId_at_a [a] (ni)
endproc (* NId_at_a *)
process NId_at_d[d](ni : NId) : noexit:=
d?tr : TPId !ni ?w : Deassign ; NId_at_d [d] (ni)
endproc (* NId_at_d *)
process NId_at_s[s](ni : NId) : noexit:=
s!ni ; NId_at_s [s] (ni)
endproc (* NId_at_s *)
process NId_at_p[p](ni : NCId) : noexit:=
p ; NId_at_p [p] (ni)
endproc (* NId_at_p *)
type TPNCIdentification
is Pair actualizedby NCEIdentification, NetworkAddress using
sortnames
    NCId for Pair
    NId for Element
    NAddress for Element2
    Bool for FBool
opnnames
    NCId for Pair
    NId for First
    RNA for Second
endtype (* TPNCIdentification *)
(*

```

## 17.3 Transfer of TPDUs

The transfer of TPDUs by means of the user data parameter of N-DATA and N-EXPEDITED DATA primitives is described by process *TPDUNSP*. The approach is similar to that followed in the LOTOS description of the end-to-end constraints of the transport service (see ISO TR 10023), in the sense of relating NSP Requests to the history of *Send* events at *p*, and *Recv* events at *p* to the history of NSP Indications. However, only NSPs that carry TPDUs are considered here, viz. N-DATA and N-EXPEDITED-DATA.

A few *NCH* subprocesses are defined later, that set constraints relating to concatenation of TPDUs (*TPDUConcatenation*, see clause 37) and to usage of the network service expedited option in class 1 (*NEXOption*, see clause 38).

```

process NCH[p, n] : noexit:=
TPDUNSP [p, n]
||
TPDUConcatenation [p, n]
||
NEXOption [p, n]
endproc (* NCH *)
(*

```

## 18 Relationship between TPDUs and NSPs

## 18.1 General

*TPDUNSP* consists of two parallel independent processes, one specifying the relationship between TPDUs and NSP per direction of transfer. The third process ignores duplicate TPDUs caused by retransmission or receipt of duplicates.

The approach is similar to that followed in the description of the end-to-end constraints in the TS formal description (see ISO TR 10023), viz.:

- SendToRequest* constrains the N-DATA requests and N-EXPEDITED-DATA requests at *n*, depending on the relevant history of *Send* events at *p*,
- IndicationToRecv* constrains the *Recv* events at *p*, depending on the relevant history of N-DATA indications and N-EXPEDITED-DATA indications.

---

```

process TPDUNSP[p, n] : noexit:=
  SendToRequestt [p, n] ||| IndicationToRecv [p, n]
endproc (* TPDUNSP *)

```

---

## 18.2 Sending of TPDUs and Requests

### 18.2.1 General

An *ETPDUHistory* describes a history of ETPDUs passed at the *p* gate. The only requirements represented here is that the order of TPDUs, for each TC assigned to the NC, must be maintained. This requirement is specified using the SRC-REF and DST-REF fields from the ETPDUs (if present).

The description of concatenation and the corresponding constraints are given in clause 37.

### 18.2.2 Process definitions

The parameter of *STREvent* represents the history of encoded TPDUs that were transferred by *Send* events at *p* but not yet transferred by a NSDU on the NC.

The alternatives in *STREvent* describe the following events:

- Sending a TPDU at the *p* gate,
- Sending a, possible concatenated, set of ETPDUs on a NC using a N-DATA request,
- Similarly, sending TPDUs on a NC using a N-EXPEDITED-DATA request.

NOTE – No internal non-determinism relating to the use or non-use of N-EXPEDITED-DATA requests is specified here.

---

```

process SendToRequestt[p, n] : noexit:=
  STREvent [p, n] (Empty of ETPDUHistory)
endproc (* SendToRequestt *)

process STREvent[p, n](psh : ETPDUHistory) : noexit:=
  p?TId : TPId ?nci : NCId ?cl : Class !Send !New ?tpdu : ETPDU ?err : TPErr ;
  STREvent [p, n] (tpdu After psh)
  []
  (choice tpdu : ETPDUHistory, nsdu : OctetString, b : Bool
  []
  [tpdu IsTCSubHistoryOf psh and (Length(tpdu) gt 0) and (nsdu eq Concatenation(tpdu))] ->
  n?NA : NAddress ?ncei : NCEI !NDTreq(nsdu, b) ; STREvent [p, n] (Remove(tpdu, psh))
  []
  (choice tpdu : ETPDUHistory, nsdu : OctetString
  []
  [tpdu IsTCSubHistoryOf psh and (Length(tpdu) gt 0) and (nsdu eq Concatenation(tpdu))] ->
  n?NA : NAddress ?ncei : NCEI !NEXreq(nsdu) ; STREvent [p, n] (Remove(tpdu, psh))
  endproc (* STREvent *)

```

---

### 18.2.3 Data definitions

*IsTCSubHistory* represents the requirement that, for a given TC, the ordering of ETPDUs may not change. It uses *TCHistory* and *TCRef*.

*TCHistory* extracts all ETPDUS from a history that belong to the same connection, according to *TPId*, as the top element of the history.

*TCRef* extracts the TC identification from an ETPDU. This is either the DST-REF or the SCR-REF in case the DST-REF equals *ZeroRef*. This function is defined in clause 40.

---

```

type SendETPDUHistory
is ETPDUHistory, TPDURferenceEncoding
opns
  _IsTCSubhistoryOf_      : ETPDUHistory, ETPDUHistory -> Bool
  _TCHistory, TCReduce   : Ref, ETPDUHistory -> ETPDUHistory
eqns
forall

```

---

## ISO/IEC TR 10024 : 1992(E)

```

    h1, h2 : ETPDUHistory, e1 : ETPDU, re : Ref
ofsort ETPDUHistory
    TCHistory(re, Empty) = Empty;
    IsETPDU(e1), Ref(e1) eq re => TCHistory(re, e1 before h1) = e1 before TCHistory(re, h1);
    IsETPDU(e1), Ref(e1) ne re => TCHistory(re, e1 before h1) = TCHistory(re, h1);
    TCReduce(re, Empty) = Empty;
    IsETPDU(e1), Ref(e1) eq re => TCReduce(re, e1 before h1) = TCReduce(re, h1);
    IsETPDU(e1), Ref(e1) ne re => TCReduce(re, e1 before h1) = e1 before TCReduce(re, h1);
ofsort Bool
    e1 before h1 IsTCSubhistoryOf h2
    = TCHistory(Ref(e1), h1) IsSubhistoryOf TCHistory(Ref(e1), h2)
    and (TCReduce(Ref(e1), h1) IsTCSubhistoryOf TCReduce(Ref(e1), h2));
endtype (* SendETPDUHistory *)
type ETPDUHistory
is History actualizedby ETPDUElement using
sortnames
    ETPDUHistory for History
    ETPDU for Element
    Bool for FBool
opnnames
    IsETPDU for IsElement
endtype (* ETPDUHistory *)
type ETPDUElement
is ETPDUCodes
opns
    IsETPDU : ETPDU -> Bool
eqns
forall
    e : ETPDU
ofsort Bool
    IsETPDU(e) = true;
endtype (* ETPDUElement *)
(*

```

To identify the connection to which an ETPDU belongs, the SRC-REF and DST-REF field are used. The reference is equal to:

- The SRC-REF from the ETPDU if the SRC-REF is not equal to *ZeroRef*;
- The DST-REF from the ETPDU if the SRC-REF equals *ZeroRef*;
- ZeroRef* otherwise.

```

type TPDUREferenceEncoding
is BasicTPDUREferenceEncoding
opns
    Ref : ETPDU -> Ref
eqns
forall
    e : ETPDU
ofsort Ref
    SrcRef(e) ne ZeroRef => Ref(e) = SrcRef(e);
    SrcRef(e) eq ZeroRef => Ref(e) = DstRef(e);
endtype (* TPDUREferenceEncoding *)
(*

```

### 18.3 Receipt of Indications and TPDUs

#### 18.3.1 General

All ETPDUs transferred in N-DATA indications or N-EXPEDITED data indications are received at the *p* gate.

#### 18.3.2 Process definitions

The parameter of *ITREvent* represents the history of ETPDUs received in N-DATA indications and N-EXPEDITED-DATA indications.

The alternatives is *ITREvent* describe the following events:

- a) Occurrence of an N-DATA indication or an N-EXPEDITED-DATA indication. The set of, possibly concatenated, TPDU's is added to the history,
- b) Receiving a TPDU at the  $p$  gate. The TPDU is removed from the history.

---

```

process IndicationToRecv[p, n] : noexit:=
  ITREvent [p, n] (Empty of etpduhistory)
endproc (* IndicationToRecv *)

process ITREvent[p, n](eh : ETPDUHistory) : noexit:=
  choice tpdus : ETPDUHistory
  []
  n?NA : NAddress ?ncei : NCEI ?ni : NSP
    [IsNDTind(ni) or IsNEXind(ni) and (UserData(ni) eq Concatenation(tpdus))] ;
    ITREvent [p, n] (tpdus After eh)
  []
  (choice t : ETPDU
    []
    [t IsFirstOf eh] ->
      p?Tid : TPID ?nci : NCID ?cl : Class !Recv !New !t ?err : TPERR ;
      ITREvent [p, n] (RemoveFirst(eh)))
  )
endproc (* ITREvent *)
  
```

---

## 19 Abstract TPDU's

### 19.1 General

A distinction is made between the definition of abstract TPDU structures, where abstraction is made from the TPDU information that relates to TC identification, encoding and checksum (in class 4), and encoded TPDU's, which are octet strings where all of the TPDU information is fully specified. TPDU information that has a mandatory value (version number in CR TPDU and EOT in ED TPDU's) are not specified in the abstract TPDU. The encoded TPDU structures (see clause 40) are endowed with suitable functions that enable one to characterize the octet strings that are valid encodings of abstract TPDU's, to relate abstract TPDU's to their valid encodings, and to access the parameter values that are not available in the abstract TPDU's, viz. references, protocol version and checksum.

In 19.2 through 19.6 the specification of abstract TPDU's is presented: the TPDU data type is built incrementally, following the same specification style that is adopted in the specification of service primitives (see clause A.2 and/or clause B.2). The description of TPDU encoding, as presented by the data type *TPDUEncoding*, is deferred to clause 40. It is to be noted that TPDU's are transferred in interactions at  $p$  in their encoded form: this is convenient to the formal description of the procedure for handling protocol errors (see clause 39).

### 19.2 Basic TPDU construction

The TPDU data type is presented starting with the basic construction of values of sort *TPDU* (see below).

The *BasicTPU* construction is derived from clause 13 of the protocol, abstracting from the encoding details and from the references, checksum, and protocol version parameters.

The functions that yield abstract TPDU values are referred to as 'TPDU constructors'. This definition imports the definitions that relate to abstract TPDU parameters (see 19.6).

---

```

type BasicTPDU
is Classes, TCAlloptions, TSAPIdentification, MaxTPDUSize, TPDUQualityParameter,
    OctetString, DRReason, ERCause, FlowControlConfirmation, TPDUNumber, Credit,
    PrMaxTPDUSize
sorts
  TPDU
opns
  CR
    : Class, TCOptions, Credit, TSAPId, TSAPId,
    MaxTPDUSize, PrMaxTPDUSize, AddOptions, Classes, TPDUQOS, OctetString -> TPDU
  CC
    : Class, TCOptions, Credit, TSAPId, TSAPId,
    MaxTPDUSize, PrMaxTPDUSize, AddOptions, TPDUQOS, OctetString -> TPDU
  DR
    : DRReason, OctetString, OctetString -> TPDU
  DC
    : -> TPDU
  DT
    : Bool, TPDUNumber, Bool, OctetString -> TPDU
  ED
    : TPDUNumber, OctetString -> TPDU
  
```

---

## ISO/IEC TR 10024 : 1992(E)

```

AK          : Credit, TPDUNumber, TPDUNumber, FCC -> TPDU
EA          : TPDUNumber                        -> TPDU
RJ          : Credit, TPDUNumber                -> TPDU
ER          : ERCause, OctetString              -> TPDU
endtype (* BasicTPDU *)

```

### 19.3 TPDU classification

A basic classification of TPDUs is defined by means of *TPDUSubsort*, which consists of a set of constants, each naming a TPDU class in one-to-one correspondence with table 8 of the protocol. The type *TPDUClassifiers* is a functional enrichment of the construction in 19.2, where:

- the classification introduced by *TPDUSubsort*,
- boolean function on TPDUs, termed 'TPDU (subsort) recognizers' are defined according to the classification introduced by *TPDUSubsort*.

```

type TPDUSubsort
is TenTuplet renamedby
sortnames
    TPDUSubsort for Tuplet
opnames
    CR for TheOne
    CC for TheOther
    DR for TheThird
    DC for TheFourth
    DT for TheFifth
    ED for TheSixth
    AK for TheSeventh
    EA for TheEighth
    RJ for TheNinth
    ER for TheTenth
endtype (* TPDUSubsort *)

type TPDUClassifiers
is BasicTPDU, TPDUSubsort
opns
    Subsort          : TPDU                        -> TPDUSubsort
    IsCR, IsCC, IsDR, IsDC, IsDT, IsED, IsAK, IsEA, IsRJ, IsER : TPDU -> Bool
eqns
forall
    cl : Class, o : TCOptions, cd : Credit, n, sn : TPDUNumber, i1, i2 : TSAPID,
    m : MaxTPDUSize, pm : PrMaxTPDUSize, ao : AddOptions, ac : Classes, q : TPDUQOS,
    u, ai, rj : OctetString, r : DRReason, roa, e : Bool, f : FCC, er : ERCause,
    tpdu : TPDU
ofsort TPDUSubsort
    Subsort(CR(cl, o, cd, i1, i2, m, pm, ao, ac, q, u)) = CR;
    Subsort(CC(cl, o, cd, i1, i2, m, pm, ao, q, u)) = CC;
    Subsort(DR(r, ai, u)) = DR;
    Subsort(DC) = DC;
    Subsort(DT(roa, n, e, u)) = DT;
    Subsort(ED(n, u)) = ED;
    Subsort(AK(cd, n, sn, f)) = AK;
    Subsort(EA(n)) = EA;
    Subsort(RJ(cd, n)) = RJ;
    Subsort(ER(er, rj)) = ER;
ofsort Bool
    IsCR(tpdu) = Subsort(tpdu) eq CR;
    IsCC(tpdu) = Subsort(tpdu) eq CC;
    IsDR(tpdu) = Subsort(tpdu) eq DR;
    IsDC(tpdu) = Subsort(tpdu) eq DC;
    IsDT(tpdu) = Subsort(tpdu) eq DT;
    IsED(tpdu) = Subsort(tpdu) eq ED;
    IsAK(tpdu) = Subsort(tpdu) eq AK;
    IsEA(tpdu) = Subsort(tpdu) eq EA;
    IsRJ(tpdu) = Subsort(tpdu) eq RJ;

```

```

IsER(tpdu) = Subsort(tpdu) eq ER;
endtype (* TPDUClassifiers *)
(*)

```

#### 19.4 TPDU parameter selectors

Boolean functions are defined, to compare or select values of parameters of TPDU's. This representation is indirect, but enables completeness of the equational definition, as well as absence of parameters depending on the value of other parameters (e.g. the class parameter in a CR or CC TPDU).

```

type TPDUPParameterSelectors
is TPDUClassifiers
opns
  _IsSelectedClassOf_, _IsPreferredClassOf_ : Class, TPDU -> Bool
  _IsOptionsOf_ : TCOptions, TPDU -> Bool
  _IsTPDUNOf_, _IsSubseqNOf_ : TPDUNumber, TPDU -> Bool
  _IsCreditOf_ : Credit, TPDU -> Bool
  _IsCallingIdOf_, _IsCalledIdOf_ : TSAPId, TPDU -> Bool
  _IsMaxSizeOf_ : MaxTPDUSize, TPDU -> Bool
  _IsPrMaxSizeOf_ : PrMaxTPDUSize, TPDU -> Bool
  _IsAddOptOf_ : AddOptions, TPDU -> Bool
  _IsAltClassesOf_ : Classes, TPDU -> Bool
  _IsTPDUQOSOf_ : TPDUQOS, TPDU -> Bool
  _IsUserDataOf_, _IsAddClearInfoOf_, _IsRejectInfoOf_ : OctetString, TPDU -> Bool
  _IsReasonOf_ : DRReason, TPDU -> Bool
  _IsEOTOf_ : Bool, TPDU -> Bool
  _IsRoAoF_ : Bool, TPDU -> Bool
  _IsERCauseOf_ : ERCause, TPDU -> Bool
  _IsFCCOf_ : FCC, TPDU -> Bool
  _IsFormatOf_ : Format, TPDU -> Bool
  _IsUseEFCOf_ : UseEFC, TPDU -> Bool
  _IsUseTEXOf_ : UseTEX, TPDU -> Bool
  _IsUseNEXOf_ : UseNEX, TPDU -> Bool
  _IsUseRCOf_ : UseRC, TPDU -> Bool
  _IsUseChecksumOf_ : UseChecksum, TPDU -> Bool
  _IsUseSackOf_ : UseSack, TPDU -> Bool
  _IsUseRackOf_ : UseRack, TPDU -> Bool
eqns
forall
  c, cl : Class, o, ol : TCOptions, cd, cd1 : Credit, n, nl, sn, sn1 : TPDUNumber,
  i1, i3, i2, i4 : TSAPID, m, ml : MaxTPDUSize, pm, pml : PrMaxTPDUSize, ao, aol : AddOptions,
  ac, acl : Classes, q, ql : TPDUQOS, u, ul, ai, ail, rj, rjl : OctetString, r, rl : DRReason,
  roa, roal, e, el : Bool, f, fl : FCC, er, erl : ERCause, fm : Format, ef : UseEFC,
  te : UseTEX, ne : UseNEX, rc : UseRC, cs : UseChecksum, p : TPDU, sack : UseSack,
  rack : UseRack
ofsort Bool
  cl IsPreferredClassOf CR(c, o, cd, i1, i2, m, pm, ao, ac, q, u) = cl eq c;
  not(IsCR(p)) => cl IsPreferredClassOf p = false;

  cl IsSelectedClassOf CC(c, o, cd, i1, i2, m, pm, ao, q, u) = cl eq c;
  not(IsCC(p)) => cl IsSelectedClassOf p = false;

  ol IsOptionsOf CR(c, o, cd, i1, i2, m, pm, ao, ac, q, u) = ol eq o;
  ol IsOptionsOf CC(c, o, cd, i1, i2, m, pm, ao, q, u) = ol eq o;
  not(IsCR(p) or IsCC(p)) => ol IsOptionsOf p = false;

  cd1 IsCreditOf CR(c, o, cd, i1, i2, m, pm, ao, ac, q, u)
    = c ge Class2 and (cd1 eq cd) or (c lt Class2 and (nat(cd) eq 0));
  cd1 IsCreditOf CC(c, o, cd, i1, i2, m, pm, ao, q, u)
    = c ge Class2 and (cd1 eq cd) or (c lt Class2 and (nat(cd) eq 0));
  cd1 IsCreditOf AK(cd, n, sn, f) = cd1 eq cd;
  cd1 IsCreditOf RJ(cd, n) = cd1 eq cd;
  not(IsCR(p) or IsCC(p) or IsAK(p) or IsRJ(p)) => cd IsCreditOf p = false;

  nl IsTPDUNOf DT(roa, n, e, u) = nl eq n;
  nl IsTPDUNOf ED(n, u) = nl eq n;

```

## ISO/IEC TR 10024 : 1992(E)

```

n1 IsTPDUNOf AK(cd, n, sn, f) = n1 eq n;
n1 IsTPDUNOf EA(n) = n1 eq n;
n1 IsTPDUNOf RJ(cd, n) = n1 eq n;
not(IsDT(p) or IsED(p) or IsAK(p) or IsEA(p) or IsRJ(p)) => n1 IsTPDUNOf p = false;

sn1 IsSubseqNOf AK(cd, n, sn, f) = sn1 eq sn;
not(IsAK(p)) => sn1 IsSubseqNOf p = false;

i3 IsCallingIdOf CR(c, o, cd, il, i2, m, pm, ao, ac, q, u) = i3 eq il;
i3 IsCallingIdOf CC(c, o, cd, il, i2, m, pm, ao, q, u) = i3 eq il;
not(IsCR(p) or IsCC(p)) => i3 IsCallingIdOf p = false;

i4 IsCalledIdOf CR(c, o, cd, il, i2, m, pm, ao, ac, q, u) = i4 eq i2;
i4 IsCalledIdOf CC(c, o, cd, il, i2, m, pm, ao, q, u) = i4 eq i2;
not(IsCR(p) or IsCC(p)) => i4 IsCalledIdOf p = false;

m1 IsMaxSizeOf CR(c, o, cd, il, i2, m, pm, ao, ac, q, u) = m1 eq m;
m1 IsMaxSizeOf CC(c, o, cd, il, i2, m, pm, ao, q, u) = m1 eq m;
not(IsCR(p) or IsCC(p)) => m1 IsMaxSizeOf p = false;

pml IsPrMaxSizeOf CR(c, o, cd, il, i2, m, pm, ao, ac, q, u) = pml eq pm;
pml IsPrMaxSizeOf CC(c, o, cd, il, i2, m, pm, ao, q, u) = pml eq pm;
not(IsCR(p) or IsCC(p)) => pml IsPrMaxSizeOf p = false;

aol IsAddOptsOf CR(c, o, cd, il, i2, m, pm, ao, ac, q, u) = aol eq ao and (c ne Class0);
aol IsAddOptsOf CC(c, o, cd, il, i2, m, pm, ao, q, u) = aol eq ao and (c ne Class0);
not(IsCR(p) or IsCC(p)) => aol IsAddOptsOf p = false;

acl IsAltClassesOf CR(c, o, cd, il, i2, m, pm, ao, ac, q, u) = acl eq ac and (c ne Class0);
not(IsCR(p)) => acl IsAltClassesOf p = false;

q1 IsTPDUQOSOf CR(c, o, cd, il, i2, m, pm, ao, ac, q, u) = q1 eq q and (c ne Class0);
q1 IsTPDUQOSOf CC(c, o, cd, il, i2, m, pm, ao, q, u) = q1 eq q and (c ne Class0);
not(IsCR(p) or IsCC(p)) => q1 IsTPDUQOSOf p = false;

ul IsUserDataOf CR(c, o, cd, il, i2, m, pm, ao, ac, q, u) = ul eq u and (c ne Class0);
ul IsUserDataOf CC(c, o, cd, il, i2, m, pm, ao, q, u) = ul eq u and (c ne Class0);
ul IsUserDataOf DR(r, ai, u) = ul eq u;
ul IsUserDataOf DT(roa, n, e, u) = ul eq u;
IsDC(p) or IsAK(p) or IsEA(p) or IsRJ(p) or IsER(p) => ul IsUserDataOf p = false;

ail IsAddClearInfoOf DR(r, ai, u) = ail eq ai;
not(IsDR(p)) => ail IsAddClearInfoOf p = false;

rjl IsRejectInfoOf ER(er, rj) = rjl eq rj;
not(IsER(p)) => rjl IsRejectInfoOf p = false;

r1 IsReasonOf DR(r, ai, u) = r1 eq r;
not(IsDR(p)) => r1 IsReasonOf p = false;

roal IsRoAOf DT(roa, n, e, u) = roal eq roal;
not(IsDT(p)) => roal IsRoAOf p = false;

el IsEOTOf DT(roa, n, e, u) = el eq e;
not(IsDT(p)) => el IsEOTOf p = false;

f1 IsFCCOf AK(cd, n, sn, f) = f1 eq f;
not(IsAK(p)) => f1 IsFCCOf p = false;

erl IsERCauseOf ER(er, rj) = erl eq er;
not(IsER(p)) => erl IsERCauseOf p = false;

fm IsFormatOf CR(c, o, cd, il, i2, m, pm, ao, ac, q, u) = fm eq Format(o);
fm IsFormatOf CC(c, o, cd, il, i2, m, pm, ao, q, u) = fm eq Format(o);
not(IsCR(p) or IsCC(p)) => fm IsFormatOf p = false;

ef IsUseEFCOf CR(c, o, cd, il, i2, m, pm, ao, ac, q, u) = ef eq UseEFC(o);
ef IsUseEFCOf CC(c, o, cd, il, i2, m, pm, ao, q, u) = ef eq UseEFC(o);

```

```

not(IsCR(p) or IsCC(p)) => ef IsUseEFCOf p = false;

te IsUseTEXOf CR(c, o, cd, i1, i2, m, pm, ao, ac, q, u) = te eq UseTEX(ao);
te IsUseTEXOf CC(c, o, cd, i1, i2, m, pm, ao, q, u) = te eq UseTEX(ao);
not(IsCR(p) or IsCC(p)) => te IsUseTEXOf p = false;

ne IsUseNEXOf CR(c, o, cd, i1, i2, m, pm, ao, ac, q, u) = ne eq UseNEX(ao);
ne IsUseNEXOf CC(c, o, cd, i1, i2, m, pm, ao, q, u) = ne eq UseNEX(ao);
not(IsCR(p) or IsCC(p)) => ne IsUseNEXOf p = false;

rc IsUserCOF CR(c, o, cd, i1, i2, m, pm, ao, ac, q, u) = rc eq UserC(ao);
rc IsUserCOF CC(c, o, cd, i1, i2, m, pm, ao, q, u) = rc eq UserC(ao);
not(IsCR(p) or IsCC(p)) => rc IsUserCOF p = false;

cs IsUseChecksumOf CR(c, o, cd, i1, i2, m, pm, ao, ac, q, u) = cs eq UseChecksum(ao);
cs IsUseChecksumOf CC(c, o, cd, i1, i2, m, pm, ao, q, u) = cs eq UseChecksum(ao);
not(IsCR(p) or IsCC(p)) => cs IsUseChecksumOf p = false;

sack IsUseSackOf CR(c, o, cd, i1, i2, m, pm, ao, ac, q, u) = sack eq UseSack(ao);
sack IsUseSackOf CC(c, o, cd, i1, i1, m, pm, ao, q, u) = sack eq UseSack(ao);
not(IsCR(p) or IsCC(p)) => sack IsUseSackOf p = false;

rack IsUserAckOf CR(c, o, cd, i1, i2, m, pm, ao, ac, q, u) = rack eq UserAck(ao);
rack IsUserAckOf CC(c, o, cd, i1, i1, m, pm, ao, q, u) = rack eq UserAck(ao);
not(IsCR(p) or IsCC(p)) => rack IsUserAckOf p = false;
endtype (* TPDUPParameterSelectors *)
(*

```

## 19.5 TPDU equality

Boolean equality on abstract TPDUs is defined as the conjunction of TPDU subsort equality (see 19.3) and pairwise equality of TPDU parameters. The latter is required only for parameters that are significant in the given TPDUs, e.g. no equality of quality of service parameters is required for CR or CC TPDUs with a Class 0 class parameter value.

The following definition completes the bottom-up part of the presentation of the abstract type TPDU. Completeness of this presentation is obtained taking also the definitions in 19.6 into account.

Other functions on TPDUs are defined close to the process definitions that make use of them, and in clause 40 (encoding).

```

type TPDU
is TPDUPParameterSelectors
opns
  _eq_, _ne_ : TPDU, TPDU -> Bool
eqns
forall c, c1 : Class, o, o1 : TCOptions, cd, cd1 : Credit, n, n1, sn, sn1 : TPDUNumber,
  i1, i3, i2, i4 : TSAPID, m, m1 : MaxTPDUSize, pm, pm1 : PrMaxTPDUSize,
  ao, aol : AddOptions, ac, ac1 : Classes, q, q1 : TPDUQOS,
  u, u1, ai, ail, rj, rj1 : OctetString, r, r1 : DRReason, roa, roal, e, e1 : Bool,
  f, f1 : FCC, er, er1 : ERCause, p, p1 : TPDU
ofsort Bool
  Subsort(p) ne Subsort(p1) => p eq p1 = false;
  c ne Class0
    => CR(c, o, cd, i1, i2, m, pm, ao, ac, q, u) eq
      CR(c1, o1, cd1, i3, i4, m1, pm1, aol, ac1, q1, u1)
  = c eq c1
    and (o eq o1)
    and (cd eq cd1)
    and (i1 eq i3)
    and (i2 eq i4)
    and (m eq m1)
    and (pm eq pm1)
    and (ao eq aol)
    and (ac eq ac1)
    and (q eq q1)
    and (u eq u1);
  CR(Class0, o, cd, i1, i2, m, pm, ao, ac, q, u) eq
    CR(c1, o1, cd1, i3, i4, m1, pm1, aol, ac1, q1, u1)

```

## ISO/IEC TR 10024 : 1992(E)

```

= c1 eq Class0
  and (i1 eq i3)
  and (i2 eq i4)
  and (m eq m1)
  and (pm eq pm1);
c ne Class0
=> CC(c, o, cd, i1, i2, m, pm, ao, q, u) eq CC(c1, o1, cd1, i3, i4, m1, pm1, ao1, q1, u1)
= c eq c1
  and (o eq o1)
  and (cd eq cd1)
  and (i1 eq i3)
  and (i2 eq i4)
  and (m eq m1)
  and (pm eq pm1)
  and (ao eq ao1)
  and (q eq q1)
  and (u eq u1);
CC(Class0, o, cd, i1, i2, m, pm, ao, q, u) eq CC(c1, o1, cd1, i3, i4, m1, pm1, ao1, q1, u1)
= c1 eq Class0
  and (i1 eq i3)
  and (i2 eq i4)
  and (m eq m1)
  and (pm eq pm1);
DR(r, ai, u) eq DR(r1, ai1, u1) = r eq r1
  and (ai eq ai1)
  and (u eq u1);
DC of TPDU eq DC = true;
DT(roa, n, e, u) eq DT(roa1, n1, e1, u1)
= n eq n1
  and (e eq e1)
  and (u eq u1)
  and (roa eq roa1);
ED(n, u) eq ED(n1, u1) = n eq n1
  and (u eq u1);
AK(cd, n, sn, f) eq AK(cd1, n1, sn1, f1)
= cd eq cd1
  and (n eq n1)
  and (sn eq sn1)
  and (f eq f1);
EA(n) eq EA(n1) = n eq n1;
RJ(cd, n) eq RJ(cd1, n1) = cd eq cd1
  and (n eq n1);
ER(er, rj) eq ER(er1, rj1) = er eq er1
  and (rj eq rj1);
p ne pl = not(p eq pl);
endtype (* TPDU *)
(*

```

## 19.6 TPDU parameters

## 19.6.1 General

TPDU parameters can be divided into five groups:

- a) Parameters relating to negotiation of options between peer entities (see 19.6.2),
- b) TSAP identification parameters (see 19.6.3),
- c) Parameters conveying information about disconnect reason and error cause (see 19.6.4),
- d) Parameters initializing flow control (see 19.6.4),
- e) Quality of Service parameters, including reassignment and acknowledgement timers (see 19.6.6).

## 19.6.2 Negotiation of options.

---

```

type TCallOptions
is TCOptions, TCAddOptions
endtype (* TCallOptions *)

type TCOptions
is Pair actualizedby Format, UseExplicitFlowControl using
sortnames
  TCOptions for Pair
  Format for Element
  UseEFC for Element2
  Bool for FBool
opnnames
  TCOptions for Pair
  Format for First
  UseEFC for Second
endtype (* TCOptions *)

type Format
is TwoTuplet renamedby
sortnames
  Format for Tuplet
opnnames
  Normal for TheOne
  Extended for TheOther
endtype (* Format *)

type UseExplicitFlowControl
is TwoTuplet renamedby
sortnames
  UseEFC for Tuplet
opnnames
  Use for TheOne
  NonUse for TheOther
endtype (* UseExplicitFlowControl *)

type TCAddOptions
is SixTuple actualizedby UseNEXC11, UserCC11, UseChecksumC14, UseTEX, UseSack, UseRack using
sortnames
  AddOptions for SixTuple
  UseNEX for Element
  UserC for Element2
  UseChecksum for Element3
  UseTEX for Element4
  UseSack for Element5
  UseRack for Element6
  Bool for FBool
opnnames
  AddOptions for Tuple
  UseNEX for First
  UserC for Second
  UseChecksum for Third
  UseTEX for Fourth
  UseSack for Fifth
  UseRack for Sixth
endtype (* TCAddOptions *)

type UseNEXC11
is TwoTuplet renamedby
sortnames
  UseNEX for Tuplet
opnnames
  Use for TheOne
  NonUse for TheOther
endtype (* UseNEXC11 *)

type UserCC11
is TwoTuplet renamedby
sortnames
  UserC for Tuplet

```

## ISO/IEC TR 10024 : 1992(E)

```

opnnames
  Use for TheOne
  NonUse for TheOther
endtype (* UserCC11 *)
type UseChecksumC14
is TwoTuplet renamedby
sortnames
  UseChecksum for Tuplet
opnnames
  Use for TheOne
  NonUse for TheOther
endtype (* UseChecksumC14 *)
type UseTEX
is TwoTuplet renamedby
sortnames
  UseTEX for Tuplet
opnnames
  Use for TheOne
  NonUse for TheOther
endtype (* UseTEX *)
type UseSack
is TwoTuplet renamedby
sortnames
  UseSack for Tuplet
opnnames
  Use for TheOne
  NonUse for TheOther
endtype (* UseSack *)
type UseRack
is TwoTuplet renamedby
sortnames
  UseRack for Tuplet
opnnames
  Use for TheOne
  NonUse for TheOther
endtype (* UseRack *)

```

(\*)

## 19.6.3 TSAP Identification

TSAPs are identified by their addresses which are of sort *TAddress*.

```

type TSAPIdentification
is TransportAddress renamedby
sortnames
  TSAPId for TAddress
endtype (* TSAPIdentification *)

```

(\*)

## 19.6.4 Disconnect reason and error cause parameters

The definition of disconnect reason is based on subclause 13.5.3 of the protocol and defines thirteen constants, together with equality and the *Applies* operation. The definition of error cause parameters is taken from subclause 13.12.2 of the protocol.

```

type DRReason
is BasicDRReason, Class
opns
  _AppliesIn_ : DRReason, Class -> Bool
eqns
forall
  drr : DRReason, cl : Class
ofsort Bool
  drr AppliesIn cl = cl ne Class0 or (h(drr) le h(AddressUnknown));
endtype (* DRReason *)

```

```

type BasicDRReason
is ThirteenTuplet renamedby
sortnames
  DRreason for Tuplet
opnnames
  Unspecified for TheOne
  TSAPCongestion for TheOther
  NoUserAtTSAP for TheThird
  AddressUnknown for TheFourth
  UserDisconnect for TheFifth
  TPECongestion for TheSixth
  NegotiationFailure for TheSeventh
  DuplicateSrcRef for TheEighth
  RefMismatch for TheNinth
  ProtocolError for TheTenth
  RefOverflow for TheEleventh
  CRRefusalOnNC for TheTwelveth
  InvalidLength for TheThirteenth
endtype (* BasicDRReason *)

```

```

type ERCause
is FourTuplet renamedby
sortnames
  ERCause for Tuplet
opnnames
  Unspecified for TheOne
  InvParCode for TheOther
  InvTPDUCode for TheThird
  InvParValue for TheFourth
endtype (* ERCause *)

```

---

#### 19.6.5 Flow control parameters

The definition a the flow control parameter is based on subclause 13.9.4 of the protocol and it describes a three tuple < *TPDUNumber* x *TPDUNumber* x *Credit* >, representing the lower window edge, subsequence number resp. credit.

---

```

type FlowControlConfirmation
is ThreeTuple actualizedby TPDUNumber, Credit using
sortnames
  FCC for ThreeTuple
  TPDUNumber for Element
  TPDUNumber for Element2
  Credit for Element3
  Bool for FBool
opnnames
  FCC for Tuple
  LWE for First
  SSN for Second
  CDT for Third
endtype (* FlowControlConfirmation *)

```

---

#### 19.6.6 Quality of Service parameters and timer values

The QOS parameter structure is decomposed according to the definitions found in items e, g, q and x of subclause 6.5.4 of the protocol. It consists of a tuple < *Protection* x *Performance* x *Timing* >. The definition of *TCProtection* is given in clause A.2.

---

```

type TPDUQualityParameter
is ThreeTuple actualizedby TCProtection, TCPerformance, TCTiming using
sortnames
  TPDUQOS for ThreeTuple
  TCProtection for Element
  TCPerformance for Element2
  TCTiming for Element3
  Bool for FBool
opnnames

```

## ISO/IEC TR 10024 : 1992(E)

```

QOS for Tuple
Protection for First
Performance for Second
Timing for Third
endtype (* TPDUQualityParameter *)
(*)

```

The definition of *TCPerformance* is similar to the one in the formal description of the Transport Service. The definition of *TCDelays* is given in clause A.2.

```

type TCPerformance
is BasicTCPerformance
opns
  _lt_, _le_, _gt_, _ge_ : TCPerformance, TCPerformance-> Bool
eqns
forall
  de, del : Delays, t, tl : Throughput, r, rl : TPRER, p, pl : TCPerformance
ofsort Bool
  Performance(de, t, r) lt Performance(del, tl, rl) = de gt del and (t lt tl) and (r gt rl);
  p le pl = p lt pl or (p eq pl);
  p gt pl = not(p le pl);
  p ge pl = not(p lt pl);
endtype (* TCPerformance *)

type BasicTCPerformance
is ThreeTuple actualizedby TCDelays, TCThroughput, TCResidualErrorRate using
sortnames
  TCPerformance for ThreeTuple
  Delays for Element
  Throughput for Element2
  TPRER for Element3
  Bool for FBool
opnnames
  Performance for Tuple
  Delays for First
  Throughput for Second
  RER for Third
endtype (* BasicTCPerformance *)
(*)

```

The definition of *TCThroughputPar* differs from the one in the formal description of the Transport Service because in the latter no distinction is made between target and minimal values.

```

type TCThroughput
is POPair actualizedby TCThroughputPar using
sortnames
  Throughput for Pair
  ThroughputPar for Element
  ThroughputPar for element2
  Bool for FBool
opnnames
  Maximum for First
  Average for Second
endtype (* TCThroughput *)

type TCThroughputPar
is DTRateDirectionQOSParameter renamedby
sortnames
  ThroughputPar for RDQOSP
opnnames
  ThroughputPar for RDQOSP
  Target for Ave
  Minimum for Max
endtype (* TCThroughputPar *)
(*)

```

A reassignment time and an acknowledgement time parameter can be present in CC or CR TPDUs. The latter parameter is used in class 4 only.

---

```

type TCTiming
is Pair actualizedby NaturalNumber using
sortnames
  TCTiming for Pair
  Nat for Element
  Nat for Element2
  Bool for FBool
opnnames
  TCTiming for Pair
  ReassTim for First
  AckTim for Second
endtype (* TCTiming *)

```

---

The definition of *TCResidualErrorRate* differs from the one in the formal description of the Transport Service because in the latter no distinction is made between target and minimal values.

---

```

type TCResidualErrorRate
is POThreeTuple actualizedby NaturalNumber, MaxTPDUSize using
sortnames
  TPRER for ThreeTuple
  Nat for Element
  Nat for Element2
  MaxTPDUSize for Element3
  Bool for FBool
opnnames
  RER for Tuple
  Target for First
  Minimum for Second
  Size for Third
endtype (* TCResidualErrorRate *)

```

---

## 20 Provision and negotiation of options

### 20.1 Provision of options

The following definition ensures that any option proposed or selected by the transport entity is actually supported - according to the value of the specification parameter *tpeo* (see clause 8 and see 8.3.2)

---

```

process SupportedOptions[p](tpeo : TPEOptions) : noexit:=
RunButSendNewCRcc [p]
[>
(choice c : TPDU
[]
  p?Tid : TPId ?nci : NCId ?cl : Class !Send !New ?etpdu : ETPDU ?err : TPErr
  [IsCR(c)or IsCC(c) and (etpdu Encodes c) and (tpeo SupportsOpts c)] ;
  Run_at_p [p])
endproc (* SupportedOptions *)

type TPEOptionSupport
is TPEOptions, TPDU, TCAlloptions
opns
  SupportsOpts_      : TPEOptions, TPDU      -> Bool
  OptionSupport      : TCOptions, AddOptions, OptionalProcedures -> Bool
eqns
forall
  c : Class, o : TCOptions, cd : Credit, i1, i2 : TSAPID, m : MaxTPDUSize,
  pm : PrMaxTPDUSize, ao : AddOptions, ac : Classes, q : TPDUQOS, u : OctetString,
  tpeo : TPEOptions, tpdu : TPDU, op : OptionalProcedures
ofsort Bool
  tpeo SupportsOpts CR(c, o, cd, i1, i2, m, pm, ao, ac, q, u)
  = c IsIn Classes(tpeo) and OptionSupport(o, ao, OptionalProcedures(tpeo))
  and (MaxTPDUSizeInClass(m, c) IsIn MaxTPDUSizeMap(tpeo))
  and (ac IsSubSetOf Classes(tpeo));

```

## ISO/IEC TR 10024 : 1992(E)

```

tpeo SupportsOpts CC(c, o, cd, i1, i2, m, pm, ao, q, u)
= c IsIn Classes(tpeo) and OptionSupport(o, ao, OptionalProcedures(tpeo))
  and (MaxTPDUSizeInClass(m, c) IsIn MaxTPDUSizeMap(tpeo));
not(IsCR(tpdu) or IsCC(tpdu)) => tpeo SupportsOpts tpdu = false;
OptionSupport(o, ao, op)
= Format(o) eq Extended implies (ExtendedFormats IsIn op)
  and (UseEFC(o) eq NonUse implies (NoFlowControl IsIn op))
  and (UseNEX(ao) eq Use implies (UseOfNSExpedited IsIn op))
  and (UseRC(ao) eq Use implies (UseOFNSReceiptConfirmation IsIn op))
  and (UseChecksum(ao) eq NonUse implies (TPDUWithoutChecksum IsIn op))
  and (UseTex(ao) eq Use implies true)
  and (UseSack(ao) eq Use implies true)
  and (UseRack(ao) eq Use implies true);
endtype (* TPEOptionSupport *)

```

(\*

## 20.2 Negotiations with the peer entity

The following definition represents the requirements found in subclause 6.5 of the protocol.

```

process TPNegotiations[p] : noexit:=
RunButNewOkCR [p]
[>
(choice ecc, ecr : ETPDU, acc, acr : TPDU
[]
[ecc Encodes acc and (ecr Encodes acr)] ->
p?Tid : TPId ?nci : NCId ?cl : Class ?d : Dir !New !ecr ?err : TPErr
[IsOK(err) and IsCR(ecr)] ;
(runButNewOkCC [p]
[>
p?Tid : TPId ?nci : NCId ?cl : Class ?d : Dir !New !ecc ?err : TPErr
[IsCC(ecc) and (not(NegotiationsOK(acr, acc)) implies HasError(err, TPNeg, InvValue))];
Run_at_p [p]))
endproc (* TPNegotiations *)

```

(\*

## 20.2.1 General

The equations relating to the factors of *NegotiationsOK* represent tables 3 and 4 of the protocol. The requirements on QOS negotiation, however, are derived from the TS definition (clause 10 of ISO 8072) under the assumption of consistency between service and protocol. These requirements are represented by the function *QOSOK*.

```

type PeerToPeerNegotiations
is ClassNegotiation, TPDUSizeNegotiation, FormatNegotiation,
FlowControlNegotiation, TExpeditedNegotiation, NExpeditedNegotiation,
NRCNegotiation, ChecksumNegotiation, QOSNegotiation, SackNegotiation,
RackNegotiation
opns
NegotiationsOK : TPDU, TPDU -> Bool
eqns
forall
cr, cc : TPDU
ofsort Bool
NegotiationsOK(cr, cc)
= ClassOK(cr, cc)
  and TPDUSizeOK(cr, cc) and FormatOK(cr, cc) and FlowControlOK(cr, cc)
  and TExpOK(cr, cc) and NExpOK(cr, cc) and NRCOK(cr, cc) and ChecksumOK(cr, cc)
  and QOSOK(cr, cc) and SackOK(cr, cc) and RackOK(cr, cc);
endtype (* PeerToPeerNegotiations *)

```

(\*

## 20.2.2 Class negotiation

The definition of *ClassNegotiation* represents the following requirements from subclause 6.5.4.j of the protocol:

*ClassOK*: The selected class must be one of the (possibly implicitly) proposed classes.

*ValidClasses*: Every alternative class must be valid according to table 3 of the protocol.

The functions *IsProposedClassesOf* and *Implicit* are used in constructing *ClassOK*.

---

```

type ClassNegotiation
is TPDU
opns
  ClassOK                : TPDU, TPDU          -> Bool
  ValidClasses           : TPDU                -> Bool
  _IsProposedClassesOf_  : Classes, TPDU        -> Bool
  Implicit               : Class               -> Classes
  Implicit               : Classes             -> Classes
eqns
forall
  tpdu, cr, cc : TPDU, c : Class, s, pc, ac : Classes
ofsort Bool
  IsCR(cr), pc IsProposedClassesOf cr, IsCC(cc), c IsSelectedClassOf cc
  => ClassOK(cr, cc) = ValidClasses(cr) and (c IsIn pc);
  not(IsCR(tpdu)) => ClassOK(tpdu, cc) = false;
  not(IsCC(tpdu)) => ClassOK(cr, tpdu) = false;
  IsCR(cr), c IsSelectedClassOf cr, pc IsProposedClassesOf cr
  => ValidClasses(cr)
  = pc Ints HigherClasses(c) eq {} and (c eq Class2 implies (Class1 NotIn pc));
  not(IsCR(tpdu)) => ValidClasses(tpdu) = false;
  IsCR(cr), c IsPreferredClassOf cr, ac IsAltClassesOf cr
  => pc IsProposedClassesOf cr = pc eq (Implicit(c) Union Implicit(ac));
  not(IsCR(tpdu)) => pc IsProposedClassesOf tpdu = false;
ofsort Classes
  Implicit(Class0) = Insert(Class0, {});
  Implicit(Class1) = Insert(Class1, Insert(Class0, {}));
  Implicit(Class2) = Insert(Class2, {});
  Implicit(Class3) = Insert(Class3, Insert(Class2, {}));
  Implicit(Class4) = Insert(Class4, Insert(Class2, {}));
  Implicit({}) = {};
  Implicit(Insert(c, s)) = Implicit(c) Union Implicit(s);
endtype (* ClassNegotiation *)

```

---

### 20.2.3 Maximum TPDU size negotiation

The definition of *TPDUSizeNegotiation* represents the following requirements from subclause 6.5.4.k of the protocol:

- The value of the *MaxTPDUSize* in a CR resp. CC TPDU must be valid for the proposed resp. selected Class; and
- The value of the *MaxTPDUSize* parameter in a CC TPDU shall not be greater than value of the *MaxTPDUSize* parameter in the preceding CR TPDU.

---

```

type TPDUSizeNegotiation
is TPDU
opns
  TPDUSizeOK                : TPDU, TPDU          -> Bool
  _IsValidMaxTPDUSizeOf_    : MaxTPDUSize, TPDU    -> Bool
  _IsValidPrMaxTPDUSizeOf_  : PrMaxTPDUSize, TPDU   -> Bool
eqns
forall
  tpdu, cr, cc : TPDU, mp, ms : MaxTPDUSize, pm, pml : PrMaxTPDUSize
ofsort Bool
  IsCR(cr), IsCC(cc), mp IsMaxSizeOf cr,
  ms IsMaxSizeOf cc,
  pm IsPrMaxSizeOf cr,
  pml IsPrMaxSizeOf cc
  => TPDUSizeOK(cr, cc)
  = mp IsValidMaxTPDUSizeOf cr
  and (ms IsValidMaxTPDUSizeOf cc)
  and (pm IsValidPrMaxTPDUSizeOf cr)
  and (pml IsValidPrMaxTPDUSizeOf cc)

```

---